Laboratorio di Intelligenza Artificiale

Elementi di Intelligenza Artificiale in Python

Luciano Capitanio

2025-07-11

Indice

Prefazione dell'autore

L'intelligenza artificiale rappresenta una delle tecnologie breakthrough più rilevanti del nostro tempo, capace di incidere in modo profondo su settori tradizionalmente fondati sull'intervento umano e sull'interpretazione, come il diritto. Dall'analisi automatica delle sentenze alla redazione assistita di contratti, dai sistemi predittivi per la valutazione del rischio di recidiva agli strumenti di supporto alla ricerca normativa, l'IA sta entrando nei contesti giuridici con crescente frequenza e rilevanza.

Questo volume nasce dall'esigenza di fornire alle studentesse e agli studenti di un corso di laurea in Giurisprudenza una prima alfabetizzazione tecnica e critica rispetto agli strumenti dell'intelligenza artificiale, con particolare riferimento al machine learning e alle sue applicazioni giuridiche. L'approccio è laboratoriale: si parte dai concetti fondamentali, illustrati con chiarezza e rigore, per poi applicarli attraverso esempi concreti in linguaggio Python, oggi uno degli strumenti più diffusi per lo sviluppo di applicazioni intelligenti.

L'obiettivo non è formare informatici, ma giuristi consapevoli, in grado di comprendere le logiche di funzionamento di un algoritmo, di interrogare criticamente le sue implicazioni normative ed etiche, e di collaborare con competenza a processi di innovazione giuridica. I temi trattati — dalla regressione alla classificazione, dalle reti neurali all'elaborazione del linguaggio naturale — sono accompagnati da

casi d'uso ed esercitazioni ispirate al contesto legale, per rendere l'apprendimento significativo e ancorato alla pratica.

Luciano Capitanio

Introduzione

L'Intelligenza Artificiale (IA) è una disciplina che si occupa della creazione di macchine e software in grado di esibire 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).

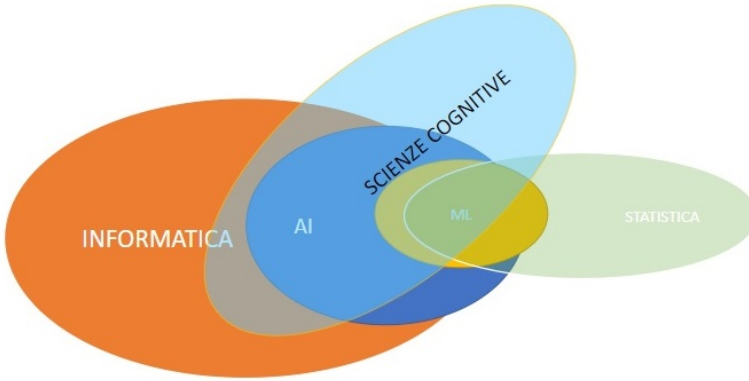


Figura 1.: L'IA è una disciplina di confine

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'intelligenza artificiale si distingue 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, senza emulare una comprensione generale. L'IA forte, invece, ipotizza sistemi in grado di possedere una coscienza e un'intelligenza paragonabili a quelle umane (Searle 1980). L'IA sta trasformando numerosi 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). Nel settore giuridico, invece, l'IA viene impiegata per automatizzare l'analisi di documenti legali, come contratti e sentenze, accelerando i processi di revisione e riducendo i costi per studi legali e aziende (Surde and Others 2024). In Italia e in Europa, la regolamentazione dell'IA è un tema centrale per bilanciare innovazione e tutela dei diritti. A livello europeo, l'Unione Europea ha introdotto l'AI Act, un quadro normativo pionieristico che classifica i sistemi di IA in base al rischio, imponendo requisiti più stringenti per applicazioni ad alto rischio, come quelle utilizzate in ambito giudiziario o nella sorveglianza. In Italia, il governo ha adottato strategie nazionali, come il Piano Strategico per l'Intelligenza Artificiale 2022-2024, per promuovere lo sviluppo responsabile dell'IA, con un focus su trasparenza, etica e protezione dei dati personali (European Parliament and Council of the European Union 2024; Governo Italiano 2021). Nonostante i progressi, l'IA presenta diverse sfide, tra cui questioni etiche come la privacy dei dati e l'impatto sull'occupazione, oltre a problemi tecnici legati alla comprensione del linguaggio naturale e allo sviluppo di algoritmi di apprendimento automatico efficaci (S. 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

1.1. Introduzione

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

Figura 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.

1.1.1. 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

Figura 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.

1.2. 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.

Parte I.

Algoritmi

2. Inferenza Logica

2.1. Introduzione

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 può essere di tre tipi :

- **inferenza deduttiva** : 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.
- **inferenza induttiva** : 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.
- **inferenza abduttiva** : 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.

2.2. 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 (DeLancey 2017). 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)

2.3. Calcolo delle Proposizioni Logiche

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

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

Tabella 2.1.: Tavola della verità per la congiunzione

p	q	p AND q
False	False	False
False	True	False
True	False	False
True	True	True

- **Disgiunzione (OR)**: L’operatore OR restituisce vero se almeno una delle due proposizioni coinvolte è vera. Ad esempio, p OR 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.

Tabella 2.2.: Tavola della verità per la disgiunzione

p	q	p OR q
False	False	False
False	True	True
True	False	True
True	True	True

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

Tabella 2.3.: Tavola della verità per la negazione

p	¬p
False	True
True	False

- **Implicazione (→)**: L'implicazione è un'operazione logica che collega due proposizioni e stabilisce una relazione di condizionalità. Si rappresenta con il simbolo "→" e si legge come "se… allora". In un'implicazione del tipo "p → 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.

Tabella 2.4.: Tavola della verità per l'implicazione

p	q	p → 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 → q". Dalla tabella della verità, possiamo vedere che in tre dei quattro casi l'implicazione "p → 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 faccia una passeggiata, in cui l'implicazione è falsa).

- **Implicazione Bilaterale (↔)**: 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 "↔" 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.

Tabella 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.

2.4. 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 (□): Se α □ β, ovvero se α implica logicamente β, in ogni mondo dove α è vera allora β è vera. È diversa dall'implicazione perché non è un connettivo logico ma una relazione che dice che se α è vera allora β è vera e basta! ## 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.

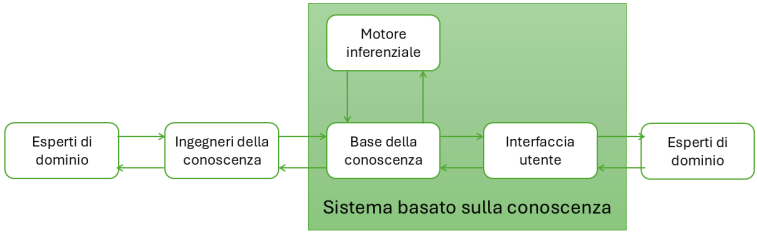


Figura 2.1.: Processo di creazione e gestione di un sistema basato sulla conoscenza

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.

Tip

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 McDermott (1980)

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.

2.5. Laboratorio Python: inferenza logica

In questo paragrafo, useremo la libreria **SymPy** dell'ecosistema 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 e per l'inferenza logica.

2.5.1. Esperimento 1: Semplice applicazione della inferenza logica al Diritto penale

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.

> **Caution**
>
> Nota: Assicurati di avere SymPy installato. Altrimenti, installalo da terminale con:
>
> ```
> pip install sympy
> ```

Glossario dei termini usati

- **Fatti**: Eventi o comportamenti specifici accaduti, rilevanti per il diritto penale in quanto potenzialmente costituenti reato o circostanze di esclusione della responsabilità.
- **Reati**: Condotte umane attive o omissive, tipiche, antigiuridiche e colpevoli, sanzionate dalla legge penale.
- **Elementi Costitutivi del Reato**: Presupposti oggettivi e soggettivi richiesti dalla norma incriminatrice per configurare un fatto come reato, tra cui la condotta (azione o omissione), il dolo o la colpa, il nesso di causalità e l'assenza di cause di giustificazione.
- **Regole Legali**: Norme giuridiche che stabiliscono i presupposti di fatto e di diritto in presenza dei quali un comportamento umano è qualificabile come reato e punibile mediante una sanzione.

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: And(Azione, Intenzione,
 ↪  NessoCausale) -> 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.

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.
```

2.5.2. Esperimento 2: Mancanza di intenzione

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.
```

2.5.3. Esperimento 3: Omicidio Colposo

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: And(Azione,
 ↪  Not(Intenzione), NessoCausale) ->
 ↪  OmicidioColposo
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.
```

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. Esplorazione che il lettore è invitato a sviluppare ulteriormente svolgendo gli esercizi proposti.

2.6. Esercizi

2.6.1. Esercizio 1: Legge di De Morgan

Verifica la seguente uguaglianza logica (Legge di De Morgan) usando le tavole della verità delle funzioni AND, NOT (¬) e OR :

```
¬(A OR B) = ¬A AND ¬B
```

2.6.2. Esercizio 2: semplice sistema esperto per valutare la legittima difesa

Implementa un sistema esperto per determinare se un caso si configura come legittima difesa utilizzando la logica proposizionale con SymPy.

La legittima difesa è ammissibile quando si verificano le seguenti condizioni:

- **Esistenza di un pericolo attuale**: Il pericolo deve essere immediato e concreto.
- **Minaccia ingiusta**: La minaccia deve essere illegittima o non giustificata.
- **Necessità di difendersi**: Deve essere impossibile evitare il pericolo in altro modo.
- **Proporzionalità della difesa**: La difesa deve essere adeguata e commisurata alla minaccia.

La codifica in Python, usando la libreria **SymPy**, potrebbe essere la seguente:

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

# Definizione dei simboli
PericoloAttuale = symbols('PericoloAttuale')
MinacciaIngiusta = symbols('MinacciaIngiusta')
NecessitaDifesa = symbols('NecessitaDifesa')
ProporzionalitaDifesa =
 ↪  symbols('ProporzionalitaDifesa')
LegittimaDifesa = symbols('LegittimaDifesa')
```

```
# Definizione della regola per la legittima difesa
regola_legittima_difesa = Implies(
    And(PericoloAttuale, MinacciaIngiusta,
 ↪  NecessitaDifesa, ProporzionalitaDifesa),
    LegittimaDifesa
)

# Caso di studio:
# Una persona reagisce a un tentativo di rapina
 ↪  armata con una spinta che fa cadere l'aggressore
fatto1 = PericoloAttuale      # C'è un rapinatore
 ↪  armato
fatto2 = MinacciaIngiusta     # La minaccia è
 ↪  ingiusta
fatto3 = NecessitaDifesa      # Non c'è via di fuga
fatto4 = ProporzionalitaDifesa # La spinta è
 ↪  proporzionata rispetto alla minaccia

# Base di conoscenza
base_conoscenza = And(fatto1, fatto2, fatto3,
 ↪  fatto4, regola_legittima_difesa)

# Verifica se si configura la legittima difesa
ipotesi = And(base_conoscenza, Not(LegittimaDifesa))
risultato = satisfiable(ipotesi)

if not risultato:
    print("Si configura la legittima difesa.")
else:
    print("Non si configura la legittima difesa.")
```

```
Si configura la legittima difesa.
```

Aggiungi una nuova regola per l'eccesso colposo di legittima difesa e Implementa l'inferenza per verificare se si configura l'eccesso di legittima difesa

2.6.3. Esercizio 3: semplice sistema esperto per valutare la premeditazione

La base di consoscenza da modificare è la seguente codificata in Python, usando la libreria **SymPy**:

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

# Definizione dei simboli
Pianificazione = symbols('Pianificazione')
AcquistoStrumenti = symbols('AcquistoStrumenti')
Appostamento = symbols('Appostamento')
SceltaVittima = symbols('SceltaVittima')
Premeditazione = symbols('Premeditazione')

# Definizione della regola per la premeditazione
regola_premeditazione = Implies(
    And(Pianificazione, AcquistoStrumenti,
 ↪  Appostamento, SceltaVittima),
    Premeditazione
)

# Caso di studio:
# Un individuo ha commesso un crimine dopo aver:
fatto1 = Pianificazione      # Pianificato l'azione
 ↪  per settimane
```

```
fatto2 = AcquistoStrumenti  # Acquistato gli
 ↪  strumenti necessari
fatto3 = Appostamento       # Studiato le abitudini
 ↪  della vittima
fatto4 = SceltaVittima      # Scelto specificamente
 ↪  la vittima

# Base di conoscenza
base_conoscenza = And(fatto1, fatto2, fatto3,
 ↪  fatto4, regola_premeditazione)

# Verifica se si configura la premeditazione
ipotesi = And(base_conoscenza, Not(Premeditazione))
risultato = satisfiable(ipotesi)

if not risultato:
    print("Si configura la premeditazione.")
else:
    print("Non si configura la premeditazione.")
```

```
Si configura la premeditazione.
```

Aggiungi una nuova regola per distinguere tra premeditazione e dolo eventuale. Implementa l'inferenza per verificare la presenza di circostanze aggravanti

3. Inferenza Probabilistica

3.1. Introduzione

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'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. 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.

3.2. Teoria delle probabilità

Il calcolo delle probabilità è una branca della matematica che misura e analizza la probabilità di eventi casuali. La probabilità rappresenta numericamente la possibilità che un evento specifico si verifichi.

> "Il concetto di probabilità è il più importante della scienza moderna, soprattutto perché nessuno ha la più pallida idea del suo significato." (Bertrand Russel) (B. Russell 1972).

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;

Ad 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**(Kolmogorov 1933).

Questi assiomi forniscono una struttura matematica rigorosa per la teoria delle probabilità, consentendo di calcolare le probabilità di eventi complessi a partire dalle probabilità di eventi più semplici:

- **Primo Assioma (Non-negatività):** La probabilità di un evento è sempre un numero reale non negativo:

  - P(A)≥0 per ogni evento A. Per rappresentare la probabilità di un certo mondo si usa il simbolo P(μ), 0 <= P(μ) <= 1
  - P(μ) = 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(μ) = 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(μ) 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 A1, A2, A3, … sono eventi mutuamente esclusivi (cioè non possono accadere simultaneamente), allora la probabilità dell'unione di questi eventi è uguale alla somma delle loro probabilità individuarie: P(A1 □ A2 □ A3 □ …) = P(A1) + P(A2) + P(A3) + … Ad esempio, la probabilità di di ottenere un numero pari nel lancio di un dado è data dalla somma delle probabilità di ottenere un 2, un 4 e un 6: P(pari) = P(2) + P(4) + P(6).

3.3. 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 nel seguente modo:

- Dal primo assioma si ha che P(x) >= 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 6x6=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 (da 2 a 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 :

Tabella 3.1.: Somma dei valori ne 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

+	1	2	3	4	5	6
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. Le probabilità di ottenere 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 sono le seguenti:

- 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

3.4. 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 {«Non luogo a procedere», «Proscioglimento», «Condanna»}
- La variabile diagnosi in campo medico può assumere uno dei valori nel dominio {«Malattia», «Non malattia»}.

3.5. 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(sentenza) = {0.1, 0.1, 0.8}. Nel seguito vedremo alcune distribuzioni di probabilità notevoli.

3.6. 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 metereologico, la probabilità che sia nuvolo e che piova è la probabilità congiunta di questi due eventi:

probabilità che sia nuvoloso:

	nuvoloso	¬nuvoloso
P(n)	0,7	0,3

probabilità che piova:

	piove	¬piove
P(p)	0,2	0,8

probabilità che sia nuvoloso e piova:

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

3.7. 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 AND b)=P(a)*P(b)

3.8. 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". Oppure, nel lancio di un dado la negazione dell'evento "esce il valore 1" è "esce il valore 2 o 3 o 4 o 5 o 6".

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

P(A) = α, allora P(¬A) = 1 - α

3.9. Inclusione

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:

P(a OR b) = P(a) + P(b) - P(a AND b).

Si noti che se gli eventi sono incompatibili la probabilità congiunta è nulla!

3.10. Marginalizzazione

La marginalizzazione è una tecnica statistica utilizzata per calcolare la probabilità complessiva di un evento integrando o sommando le probabilità congiunte rispetto a un insieme di stati possibili. Ad esempio, consideriamo l'evento a (un imputato è condannato) e gli eventi mutuamente esclusivi b (l'imputato è colpevole) e ¬b (l'imputato non è colpevole). La probabilità totale di a si calcola come:

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

dove P(a, b) è la probabilità che a si verifichi dato che b si verifica, e P(a, ¬b) è la probabilità che a si verifichi dato che ¬b si verifica.

Questa somma considera entrambi gli stati possibili del mondo (b e ¬b) per calcolare la probabilità totale di condanna. Poiché b e ¬b sono mutuamente esclusivi, le loro probabilità possono essere sommate senza sovrapposizioni.

3.11. 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?

Ad esempio, la probabilità che un imputato sia condannato dato che è colpevole è diversa dalla probabilità che un imputato sia condannato.

La probabilità condizionata è la probabilità che un evento si verifichi dato che un altro evento si è verificato.

È 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.

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".

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

$$P(a|b) = P(a,b)/P(b);$$

"siamo interessati agli eventi dove a e b sono vere, ma solo nei mondi dove b è vera!" Dalla formula precedente discendono immediatamente le seguenti due formule per il calcolo della probabilità congiunta:

$$P(a,b) = P(b)P(a|b);$$
$$P(a,b) = P(a)P(b|a);$$

3.12. Condizionamento

Il condizionamento discende immediatamente dalla marginalizzazione. La probabilità che si verifichi a è data dalla marginalizzazione della probabilità congiunta di questi due eventi.

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

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

3.13. Laboratorio Python: inferenza probabilistica

3.13.1. Esperimento 1: Variabili aleatorie

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(sentenza = «Non luogo a procedere») = 0,1
- P(sentenza = «Proscioglimento») = 0,1
- P(sentenza = «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
```

3.13.2. Esperimento 2: Probabilità condizionate

Posto che si hanno le seguenti variabili aleatorie: - a = condanna - b = prova schiacciante

e che si hanno le seguenti informazioni raccolte in un contesto giuridico penale:

- P(a|b) = 0.95
- P(a|¬b) = 0.2
- P(b) = 0.3

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

P(a)= P(a|b)P(b) + P(a|¬b)P(¬b);

La codifica in Python è la seguente:

```
Pa_cond_b    = 0.95
Pa_cond_nonb = 0.2
Pb = 0.3
Pnonb = 1 - Pb
Pa = Pa_cond_b * Pb + Pa_cond_nonb * Pnonb
print(f"La probabilità che un imputato sia colpevole
 ↪  dato che ci sono prove schiaccianti è: {Pa}")
```

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

3.13.3. Esperimento 3: Probabilità congiunta

Supponiamo di avere le seguenti variabili aleatorie: - a = condanna - b = prova schiacciante

e che si hanno le seguenti informazioni raccolte in un contesto giuridico penale: - P(a) = 0.8 - P(b) = 0.3 - P(a|b) = 0.95

Per calcolare la probabilità congiunta di a,b si utilizza la formula: P(a,b) = P(a|b)P(b) La codifica in Python è la seguente:

```
Pa = 0.8
Pb = 0.3
Pa_cond_b = 0.95
Pab = Pa_cond_b * Pb
print(f"La probabilità congiunta di a,b è: {Pab}")
```

```
La probabilità congiunta di a,b è: 0.285
```

3.14. Esercizi

3.14.1. Esercizio 1: saturazione di una variabile aleatoria

Consideriamo l'evento a (un imputato è condannato) e gli eventi mutuamente esclusivi b (l'imputato è colpevole) e ¬b (l'imputato non è colpevole). Supponiamo di avere i seguenti valori:

- P(a,b) = 0.9
- P(a,¬b) = 0.2

Si calocoli la probabilità di essere condannato P(a).

3.14.2. Esercizio 2: probabilità condizionata

I risultati che si ottengono dall'esperimento di lancio di un dado sono i seguenti:

Tabella 3.5.: 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

Qual è la probabilità che si ottenga una somma pari 9 lanciando due dadi se il primo dado è 6, P(9/6)?

1. Si calcoli tale probabilità condizionata utilizzando la definizione di probabilità condizionata : P(a|b) = P(a,b)/P(b).
2. Si ripeta il calcolo applicando la definizione di probabilità. Ovvero, la probabilità di un certo evento è data dal rapporto tra il numero di casi favorevoli e il numero di casi possibili.

3.14.3. Esercizio 3: probabilità congiunta

Supponiamo di avere raccolto informazioni su 1.000 casi penali. Per ogni caso, sono state raccolte le seguenti informazioni:

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

Il dataset (l'insieme dei dati raccolti) è il seguente:

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**

Si calcoli:

- la probabilità che un sospettato sia condannato dato che non ha un alibi e c'è un testimone oculare.

- la probabilità che ci sia un testimone oculare dato che il sospettato è stato condannato.

4. Inferenza Bayesiana

4.1. Introduzione

> "Mr. Bayes … design … was to find out a method by which we might judge concerning the probability that an event has to happen, in dato 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.

4.2. 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.

4.3. 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 informazioni (false perché inventate dall' autore :) ottenute da un ipoteticodi un certo tribunale o dal Ministero della Giustizia abbiamo che

- 80% degli imputati condannati hanno precedenti penali P(precedenti/condanna) = 0,8;
- 10% degli imputati sono condannati P(condanna) = 0,1
- 20% degli imputati hanno precedenti penali P(precedenti) = 0,2

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

$$P(cond/prec) = P(cond) \cdot \frac{P(prec/cond)}{P(prec)} = 0,1 \cdot \frac{0,8}{0,2} = 0,4$$

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 cosidetto 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.

4.4. 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 sintomi e malattie. Dati i sintomi, la rete può essere usata per calcolare la probabilità della presenza di diverse malattie. Ogni nodo della BN rappresenta una variabile aleatoria e ogni freccia dal nodo X al nodo 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 | Genitori(X)). La definizione della topologia di una BN o rete di credenza è affidata ad un esperto di dominio che stabilisce quali nodi e quali relazioni condizionali di dipendenza sono utili per modellare gli eventi del problema in esame. 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, gli archi della rete, riassumono un insieme potenzialmente infinito di circostanze a noi ignote e che potrebbero influenzare l'evento esprimendo relazioni causali dirette (causa -> 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.

Ad esempio, la seguente rete bayesiana rappresenta la relazione tra gli eventi che possono condizionare l'arrivo puntuale ad un appuntamento usando un mezzo di trasporto come ad esempio un treno.

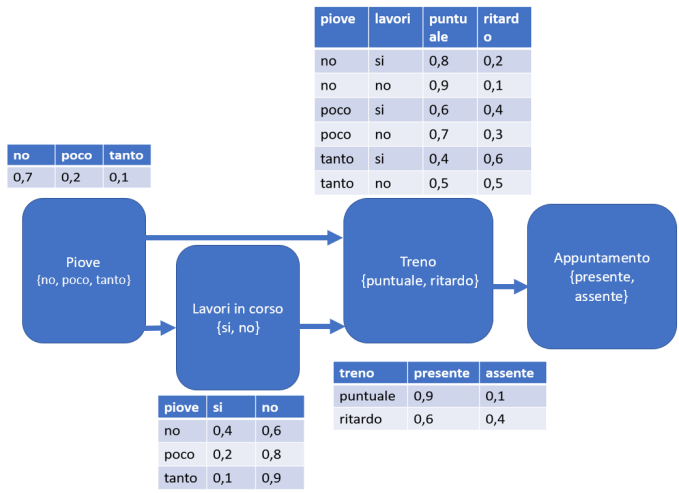


Figura 4.1.: rete di Bayes appuntamento

4.5. laboratorio di Python

4.5.1. Esperimento 1 - Inferenza Bayesiana

La formula di Bayes descrive la probabilità condizionata di un evento A dato un evento B. La formula è la seguente:

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

Il denominatore P(B) può essere calcolato nel seguente modo:

$$P(B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A)$$

Una possibile implementazione in Python della formula di Bayes è la seguente:

```
def bayes_theorem(p_a, p_b_dato_a, p_b_dato_not_a):
  # Calcola la probabilità di B dato A
  p_a_dato_b = (p_b_dato_a * p_a) / (p_b_dato_a *
 ↪  p_a + p_b_dato_not_a * (1 - p_a))
  return p_a_dato_b
```

Proviamo ad applicare la formula a un semplice caso penale. Immaginiamo di avere le seguenti informazioni:

- P(condanna) = 0.1;
- P(precedenti|condanna) = 0.8;
- P(precedenti|non condanna) = 0.1;

Calcoliamo la probabilità di condanna dato che ci sono precedenti penali.

```
# Esempio di applicazione al caso penale
# Probabilità a priori
p_precedenti = 0.2  # P(precedenti)
p_condanna = 0.1    # P(condanna)
p_precedenti_dato_condanna = 0.8  #
 ↪  P(precedenti|condanna)

# Calcolo della probabilità di condanna dato che ci
 ↪  sono precedenti
p_condanna_dato_precedenti = bayes_theorem(
    p_a=p_condanna,
    p_b_dato_a=p_precedenti_dato_condanna,
    p_b_dato_not_a=0.1  # P(precedenti|non condanna)
)
fattore_bayes = p_condanna_dato_precedenti /
 ↪  p_condanna
# Stampa i risultati
print(f"La probabilità di condanna a priori è:
 ↪  {p_condanna:.1%}")
print(f"La probabilità di condanna dato che ci sono
 ↪  precedenti penali è:
 ↪  {p_condanna_dato_precedenti:.1%}")
print(f"Il fattore di Bayes è: {fattore_bayes:.2f}")
print(f"Quindi il fatto di sapere che l'imputato ha
 ↪  precedenti penali moltiplica di
 ↪  {fattore_bayes:.2f} volte la probabilità di
 ↪  condanna!")
```

```
La probabilità di condanna a priori è: 10.0%
La probabilità di condanna dato che ci sono
precedenti penali è: 47.1%
Il fattore di Bayes è: 4.71
```

```
Quindi il fatto di sapere che l'imputato ha
precedenti penali moltiplica di 4.71 volte la
probabilità di condanna!
```

4.5.2. Esperimento 2 - Reti bayesiane

Scriviamo insieme il codice Python necessario per creare un modello di Rete Bayesiana per analizzare la probabilità di colpevolezza di un sospetto in un'indagine criminale. Tenendo conto del fatto che i modelli sono sempre un'approssimazione del mondo reale, si può utilizzare la teoria delle reti di Bayes per modellare la probabilità di un evento in base a vari fattori di prova. Il modello considera solo tre elementi di prova: la presenza di un'arma (Arma), un movente (Movente) 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, Movente, 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	Motivo	Alibi	P(Non Colpevole)	P(Colpevole)
0	0	0	0.9	0.1
0	0	1	0.99	0.01
0	1	0	0.7	0.3
0	1	1	0.79	0.21
1	0	0	0.5	0.5
1	0	1	0.59	0.41
1	1	0	0.2	0.8
1	1	1	0.1	0.9

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 inquirenti a quantificare e ragionare sull'incertezza nei casi criminali.

> **Caution**
>
> Se non abbiamo mai installato le librerie pgmpy, networkx e matplotlib, l'interprete Python segnalerà che pgmpy non è riconosciuta. In questo caso è necessario installare le librerie necessarie eseguendo il seguente blocco di codice:
>
> `!pip install pgmpy networkx matplotlib`

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

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

# Definizione delle probabilità condizionate
cpd_arma = TabularCPD(variable='Arma',
 ↪  variable_card=2,
                       values=[[0.7], [0.3]])
cpd_Movente = TabularCPD(variable='Movente',
 ↪  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=[[0.1, 0.01,
 ↪  0.3, 0.21, 0.5, 0.41, 0.8, 0.9],
                                        [0.9, 0.99,
 ↪  0.7, 0.79, 0.5, 0.59, 0.2, 0.1]],
                                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)
```

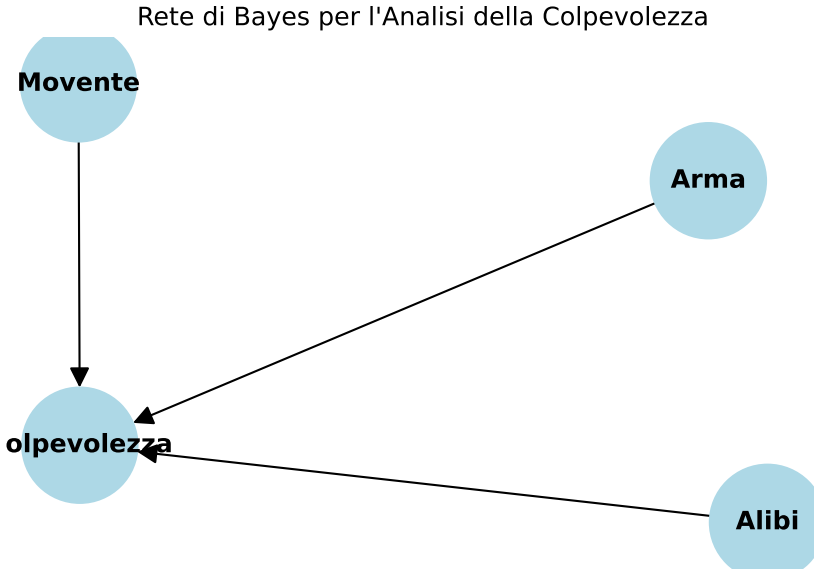
```
Il modello è corretto:  True
CPD di Arma:
+---------+-----+
| Arma(0) | 0.7 |
+---------+-----+
| Arma(1) | 0.3 |
+---------+-----+
CPD di Movente:
+------------+-----+
| Movente(0) | 0.6 |
```

```
+------------+-----+
| 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) | 0.1        | ... | 0.8        |
0.9        |
+-----------------+------------+-----+------------+------------
| Colpevolezza(1) | 0.9        | ... | 0.2        |
0.1        |
+-----------------+------------+-----+------------+------------
```

Possiamo anche visualizzare la rete di Bayes con il seguente codice:

```
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()
```



Infine, una volta costruita la rete di Bayes possiamo interrogarla per avere una stima della probabilità di un determinato evento. Qual è la

probabiltà 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[0])
print("Probabilità di innocenza:")
print(result.values[1])
```

```
Probabilità di colpevolezza:
0.1
Probabilità di innocenza:
0.9
```

```
from pgmpy.inference import VariableElimination

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

# Definiamo l'evidenza per la situazione descritta
evidence = {
    'Alibi': 1,   # 1 rappresenta la presenza 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[0])
print("Probabilità di innocenza:")
print(result.values[1])
```

```
Probabilità di colpevolezza:
0.01
Probabilità di innocenza:
0.99
```

4.6. Esercizi

4.6.1. Esercizio 1: teorema di Bayes

Supponiamo di avere raccolto informazioni su 1.000 casi penali. Per ogni caso, sono state raccolte le seguenti informazioni:

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

Il dataset (l'insieme dei dati raccolti) è il seguente:

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**

Si vuole comprendere l'impatto della assenza di un alibi sulla probabilità di condanna. A questo scopo, si applichi il teorema di Bayes per calcolare la probabilità che un sospettato venga **condannato dato che non ha un alibi**, cioè $P(\text{Condanna} = \text{Sì} \mid \text{Alibi} = \text{No})$. Per comodità si riporta qui di seguito la formula del teorema di Bayes applicata a questo caso specifico:

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

Ovvero, per risolvere l'esercizio occorre trovare i seguenti valori:

- $P(\text{Alibi} = \text{No} \mid \text{Cond} = \text{Sì})$
- $P(\text{Cond} = \text{Sì})$
- $P(\text{Alibi} = \text{No})$

4.6.2. Esercizio 2: Fattore di Bayes

Si calcoli il fattore di Bayes per l'inferenza Bayesiana studiata nell'esercizio 1.

4.6.3. Esercizio 3: Probabilità a posteriori

Si calcoli la probabilità che un sospettato venga condannato dato che non ha un alibi e che non ha un testimone oculare.

5. Algoritmi di Ricerca

5.1. Introduzione

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.

I problemi che si possono affrontare con gli algoritmi di ricerca sono quei problemi dove:

Un agente che opera in un certo ambiente si trova in uno **Stato iniziale** (Questo rappresenta la condizione o la configurazione di partenza del problema). Esso, mediante le sue **Azioni** (le possibili mosse o le transizioni che possono essere effettuate a partire da uno stato) cerca di raggiungere lo **stato finale** o **obiettivo**. Il raggiungimento dello stato obiettivo è misurabile mediante un **Test obiettivo (goal test)** (un criterio che determina se uno stato specifico risolve il problema). Ogni azione o operatore dell'agente ha un costo associato, che può essere costante o variabile a seconda della natura del problema, che può essere misurato mediante una **Funzione costo** (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 obiettivo[1].

esempio di problema di ricerca

Un esempio classico di problema che segue questa struttura è 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.

5.2. 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**. 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 (S. J. Russell and Norvig 2021):

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

L'algoritmo di ricerca generale appena descritto è un approccio generale per risolvere problemi di ricerca. Tuttavia, per risolvere un problema specifico, è necessario specificare una strategia di ricerca. Una strategia di ricerca è una regola che determina l'ordine in cui i nodi vengono esplorati nella frontiera. Le strategie di ricerca possono essere classificate in base a diversi criteri, tra cui: 1. **Strategie di ricerca non informate**: Queste strategie non utilizzano alcuna informazione aggiuntiva oltre a quella fornita dal problema stesso. 2. **Strategie di ricerca informate**: Queste strategie utilizzano informazioni aggiuntive per guidare la ricerca. 3. **Strategie di ricerca a costo minimo**: Queste strategie utilizzano un costo per guidare la ricerca.

5.3. Algoritmi di ricerca non informati

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. La scelta del nodo da rimuovere dalla frontiera è basata su una strategia di ricerca, che determina l'ordine in cui i nodi vengono esplorati. Nel laboratorio di Python alla fine di questo capitolo vedremo come si implementano le strategie di ricerca più comuni. Invece, qui di seguito ipotizziamo di voler trovare il percorso più breve tra i nodi A e F del seguente grafo:

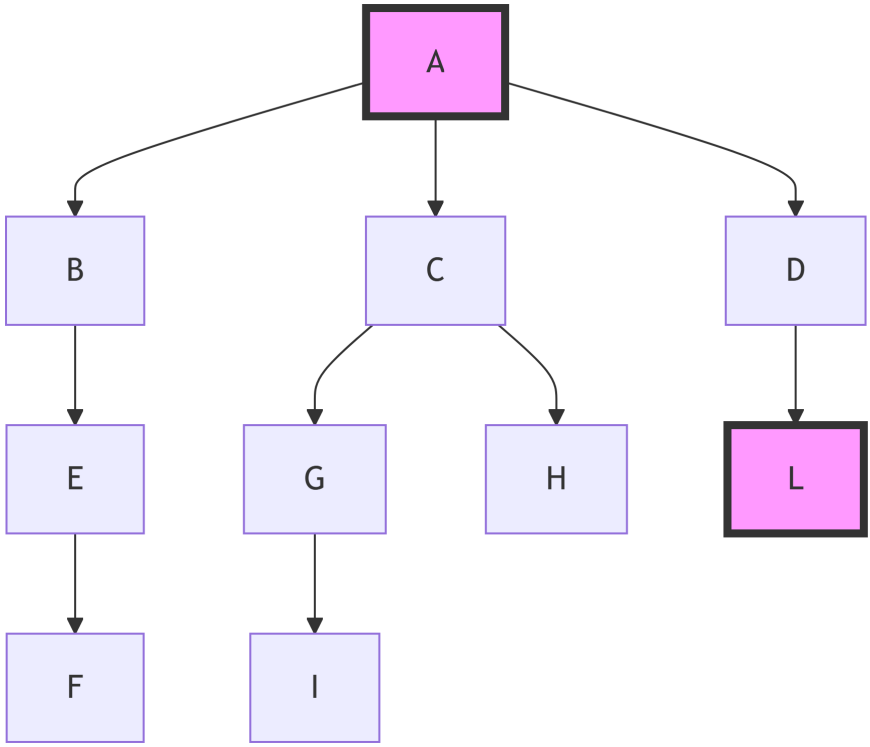


Figura 5.1.: Grafo dove cercare il percorso tra il nodo A e il nodo L.

Nel fare questa ricerca usiamo due strutture per la frontiera:

- Lo **pila**: L'ultimo nodo inserito è il primo estratto (LIFO = Last In First Out) come accade per una **pila** di piatti dove il più in alto è il primo a essere prelevato.
- La **coda**: Il primo nodo inserito è il primo estratto (FIFO = First In First Out) come accade per una **coda** di persone ad uno sportello dove la prima persona in coda è la prima a essere servita.

Se applichiamo l'algoritmo generale di ricerca al nostro grafo, utilizzando la strategia di ricerca a pila, o LIFO (Last In, First Out), il nodo più recentemente aggiunto alla frontiera è il primo a essere esplorato. Questo comporta una **ricerca in profondità** nota come **DFS - Depth First Search**:

- L'algoritmo esplora un ramo del grafo fino a raggiungere un vicolo cieco o la soluzione.
- Se non si trova la soluzione, torna indietro per esplorare altri rami.

Nella tabella seguente, mostriamo i passi dell'esecuzione dell'algoritmo con il nodo candidato, il contenuto della frontiera, i nodi visitati e il percorso candidato per ogni passo dell'algoritmo:

Tabella 5.1.: ricerca del percorso tra i nodi A e L utilizzando la strategia di ricerca a pila

Passo	Nodo Candidato	Frontiera	Nodi Visitati	Percorso
1	A	[D, C, B]	[A]	[A]
2	B	[D, C, E]	[A, B]	[A, B]
3	E	[D, C, F]	[A, B, E]	[A, B, E]
4	F	[D, C]	[A, B, E, F]	[A, B, E, F]
5	C	[D, H, G]	[A, B, E, F, C]	[A, C]
6	G	[D, H, I]	[A, B, E, F, C, G]	[A, C, G]
7	I	[D, H]	[A, B, E, F, C, G, I]	[A, C, G, I]
8	H	[D]	[A, B, E, F, C, G, I, H]	[A, C, H]
9	D	[L]	[A, B, E, F, C, G, I, H, D]	[A, D]

Passo	Nodo Candidato	Frontiera	Nodi Visitati	Percorso
10	L	[]	[A, B, E, F, C, G, I, H, D, L]	[A, D, L]

L'esito della ricerca DFS è mostrato nel seguente grafo dove i nodi visitati sono evidenziati in verde e numerati in ordine di visita e il percorso trovato ha i rami di colore rosso:

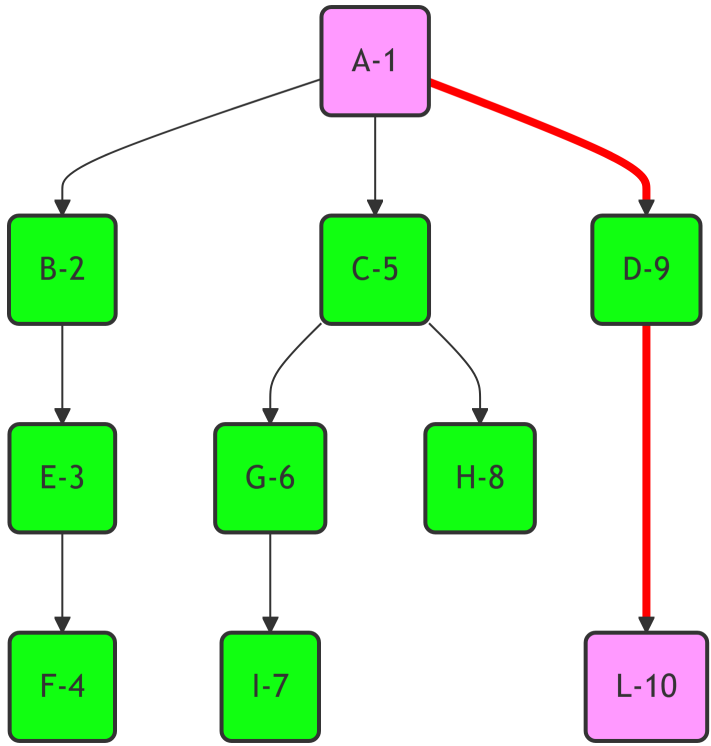


Figura 5.2.: Esito della ricerca DFS: Percorso trovato: [A, D, L].

Se applichiamo l'algoritmo generale di ricerca al nostro grafo, utilizzando la strategia di ricerca a coda, o FIFO (First In, First Out), il nodo più vecchio è il primo a essere esplorato. Questo comporta una **ricerca in ampiezza** nota come **BFS - Breath First Search**:

- L'algoritmo esplora i nodi vicini prima di passare a quelli più lontani.
- Trova sempre il percorso più breve (in termini di numero di nodi).

Nella tabella seguente, mostriamo i passi dell'esecuzione dell'algoritmo con il nodo candidato, il contenuto della frontiera, i nodi visitati e il percorso candidato per ogni passo dell'algoritmo:

Tabella 5.2.: ricerca del percorso tra i nodi A e L utilizzando la strategia di ricerca a coda

Passo	Nodo Candidato	Frontiera	Nodi Visitati	Percorso
1	A	[B, C, D]	[A]	[A]
2	B	[C, D, E]	[A, B]	[A, B]
3	C	[D, E, G, H]	[A, B, C]	[A, C]
4	D	[E, G, H, L]	[A, B, C, D]	[A, D]
5	L	[E, G, H]	[A, B, C, D, L]	[A, D, L]

L'esito della ricerca BFS è mostrato nel seguente grafo dove i nodi visitati sono evidenziati in verde e numerati in ordine di visita e il percorso trovato ha i rami di colore rosso:

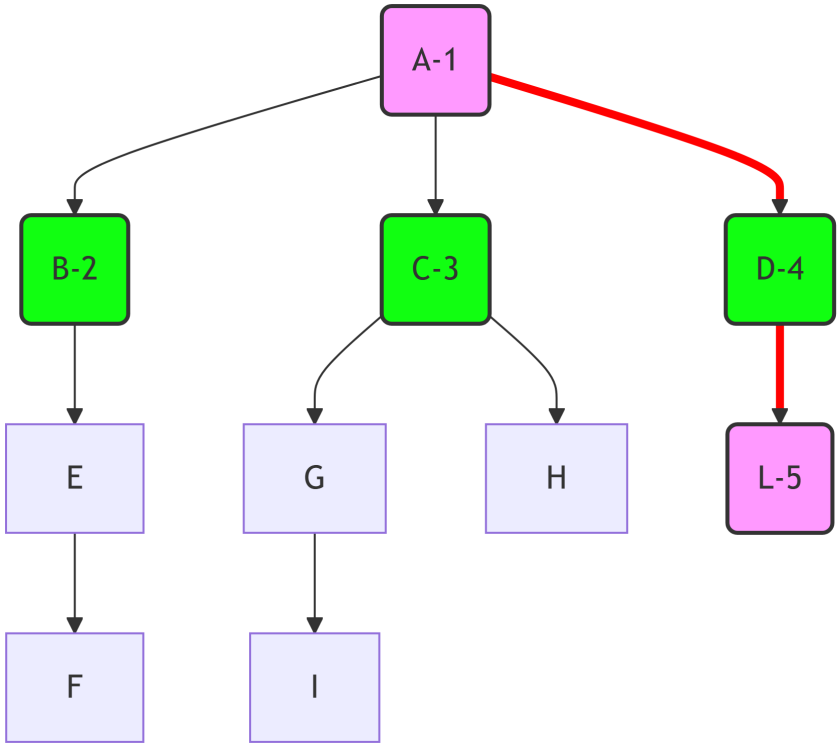


Figura 5.3.: Esito della ricerca BFS: Percorso trovato: [A, D, L].

Confrontando i due algoritmi di ricerca, possiamo notare che:

1. **Pila (DFS)**:
    - Esplora un ramo alla volta.
    - Potrebbe visitare più nodi rispetto alla soluzione ottimale.
2. **Coda (BFS)**:
    - Trova sempre il percorso più breve in termini di numero di nodi.

- Esplora i nodi livello per livello.

In generale, la scelta tra DFS e BFS dipende dal problema specifico e dalle caratteristiche del grafo.

5.4. Algoritmi di Ricerca Informati

Gli **algoritmi di ricerca informati** sono una classe di algoritmi che utilizzano informazioni aggiuntive, chiamate **euristiche**, per guidare il processo di ricerca verso la soluzione in modo più efficiente rispetto agli approcci non informati. L'**euristica** è una funzione che stima il costo o la distanza dal nodo corrente al nodo obiettivo, fornendo un'indicazione di "quanto vicino" ci si trovi alla soluzione. 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.

5.4.1. 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. Le euristiche dovrebbero essere **ammissibili** e **consistenti**:

1. **Euristiche ammissibili**:
    - Una funzione euristica è ammissibile se non sovrastima mai il costo reale per raggiungere l'obiettivo, garantendo che l'algoritmo trovi il percorso ottimale.
    - Ad esempio, in un problema di navigazione, la distanza in linea retta tra due punti è un'euristica ammissibile.
2. **Euristiche consistenti (o monotone)**:
    - Una funzione euristica è consistente se per ogni nodo n, il costo stimato dall'euristica soddisfa la disuguaglianza del triangolo:

$$h(n) \leq c(n, n') + h(n')$$

Dove $c(n, n')$ è il costo effettivo tra n e n'. Questa proprietà garantisce che A* non rientri mai in un nodo già visitato. La progettazione di funzioni euristiche efficaci è spesso una sfida cruciale nell'applicazione degli algoritmi di ricerca informati a problemi complessi del mondo reale.

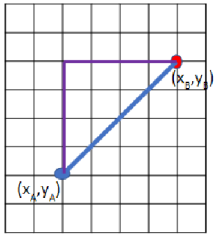


Figura 5.4.: Distanze euclidee e di Manhattan

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:

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

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 del 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.

Numero di pezzi fuori posto: Usata nei giochi di puzzle (es. il gioco del 15), conta il numero di tessere non in posizione corretta.

5.4.2. Greedy Best-First Search

L'algoritmo utilizza una **priority queue**, una coda che mantiene i nodi in ordine di priorità, per gestire la frontiera. In questo algoritmo la priorità è determinata unicamente dalla funzione euristica $h(n)$, che stima quanto il nodo n sia vicino all'obiettivo. Funzionamento:

1. Si inseriscono nella frontiera i nodi vicini al nodo corrente.
2. Si estrae il nodo con il valore di $h(n)$ più basso.
3. Si ripete fino a raggiungere l'obiettivo o a esaurire i nodi.

Questo algoritmo è efficiente, ma non garantisce la soluzione ottimale, poiché si basa solo sull'euristica.

Nel laboratorio di Python alla fine di questo capitolo vedremo come si implementa questa strategia di ricerca. Qui di seguito

ipotizziamo di voler trovare il percorso più breve tra i nodi due località : Castelnuovo di Porto(RM) e Viale Pola, 12, Roma. Il grafo semplificato è il seguente:

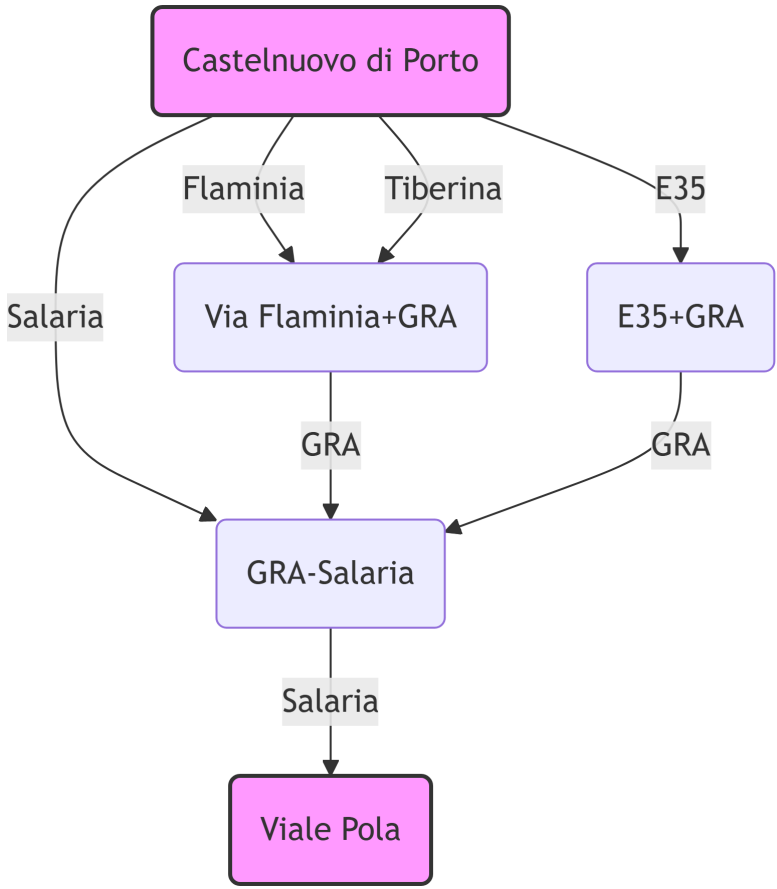


Figura 5.5.: Esempio di grafo per le funzioni di ricerca informate

L'eurstica utilizzata è la distanza euclidea tra i nodi, le località, che è una stima ammissibile:

località	distanza da Via Pola
A - Castelnuovo di Porto	23
B - Via Flaminia+GRA	7
C - E35+GRA	7
D - GRA-Salaria	8
E - Viale Pola	0

Nella tabella seguente, mostriamo i passi dell'esecuzione dell'algoritmo con il nodo candidato, il contenuto della frontiera, i nodi visitati, il percorso candidato e l'euristica per ogni passo dell'algoritmo:

Tabella 5.4.: ricerca del percorso tra i nodi A e L utilizzando l'algoritmo Greedy Best-First Search.

Passo	Nodo Candidato	Frontiera	Nodi Visitati	Percorso	Euristica
1	A	[(B, 7), (C, 7), (D, 8)]	[A]	[A]	23
2	B	[(C, 7), (D, 8), (D, 8)]	[A, B]	[A, B]	7
3	C	[(D, 8), (D, 8)]	[A, B, C]	[A, C]	7
4	D	[(D, 8)]	[A, B, C, D]	[A, D]	8
5	E	[]	[A, B, C, D, E]	[A, D, E]	0

Spiegazione dei Passaggi

1. **Passo 1**:

    - Nodo iniziale A viene visitato.
    - I vicini B, C, D sono aggiunti alla frontiera con le rispettive euristiche.
    - Nodo candidato successivo: B (euristica più bassa).

2. **Passo 2**:

    - Nodo B viene estratto dalla frontiera e visitato.
    - Vicino D viene aggiunto di nuovo alla frontiera, ma ha già un'euristica assegnata.
    - Nodo candidato successivo: C (euristica più bassa).

3. **Passo 3**:

    - Nodo C viene estratto e visitato.
    - Il vicino D non cambia la priorità della frontiera.
    - Nodo candidato successivo: D (euristica più bassa).

4. **Passo 4**:

    - Nodo D viene estratto e visitato.
    - Il vicino E è aggiunto alla frontiera.
    - Nodo candidato successivo: E (obiettivo raggiunto).

5. **Passo 5**:

    - Nodo E viene estratto e visitato.
    - Il nodo obiettivo è stato raggiunto, e l'algoritmo termina.

Percorso trovato: $[A, D, E]$
Euristica utilizzata: La scelta dei nodi è basata esclusivamente sul valore $h(n)$, senza considerare il costo effettivo del cammino. Quindi l'algoritmo usato senza altre informazioni oltre alla distanza da Viale Pola indica di percorrere un tratto della salaria fino al GRA e continuare sulla Salaria per raggiungere Viale Pola.

5.4.3. A* (A-Star Search)

L'algoritmo utilizza una **priority queue** con una funzione di priorità combinata:

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

Dove:

- $g(n)$: costo del percorso dal nodo iniziale al nodo n.
- $h(n)$: stima del costo per raggiungere l'obiettivo da n.

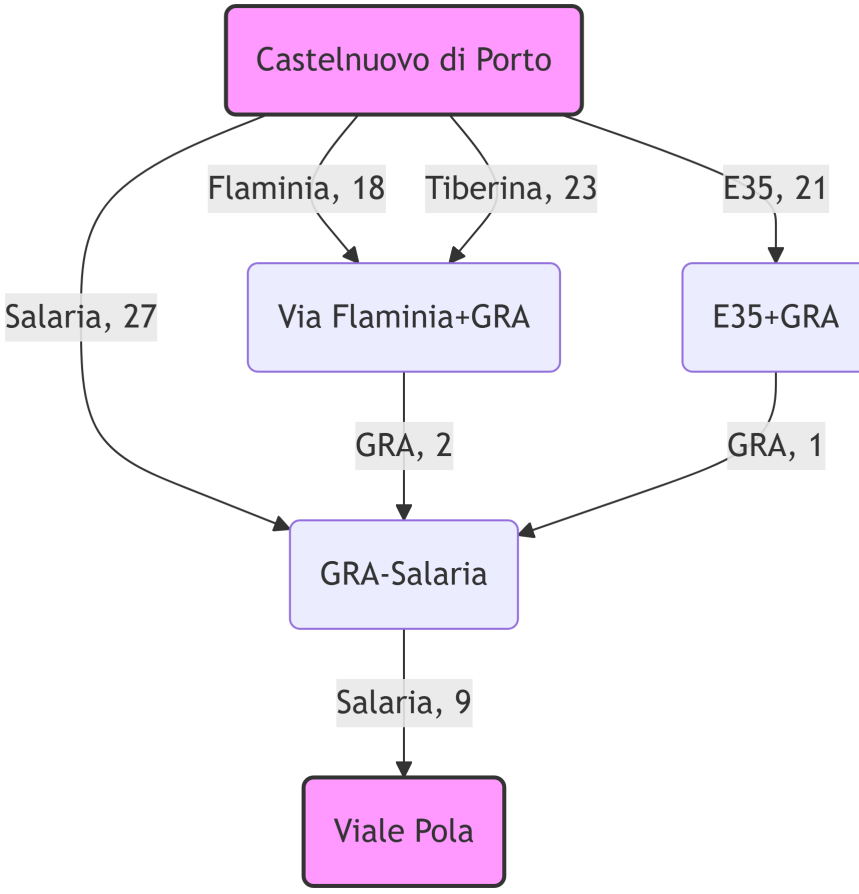
Funzionamento:

1. Ogni nodo inserito nella frontiera ha una priorità basata su $f(n)$.
2. Si estrae il nodo con il valore di $f(n)$ più basso.
3. L'algoritmo continua fino a trovare il nodo obiettivo con il costo più basso.

Differenza tra Greedy e A*:

- **Greedy Best-First Search** si concentra solo su* $h(n)$ (euristica).
- **A*** bilancia $g(n)$ (costo effettivo) e $h(n)$ (euristica), garantendo una soluzione ottimale se l'euristica è ammissibile e consistente.

Proviamo ad applicare l'algoritmo A* al calcolo del percorso più breve nel caso dell'esempio visto nel paragrafo precedente. Riportiamo qui di seguito il grafo semplificato dove sono riportati anche i valori di costo stimati usando la distanza su strada tra i nodi del grafo:



L'eurstica utilizzata è la distanza euclidea tra i nodi, le località, che è una stima ammissibile come nel paragrafo precedente:

località	distanza da Via Pola
A - Castelnuovo di Porto	23
B - Via Flaminia+GRA	7
C - E35+GRA	7
D - GRA-Salaria	8
E - Viale Pola	0

località	distanza da Via Pola

Nella tabella seguente, mostriamo i passi dell'esecuzione dell'algoritmo con il nodo candidato, il contenuto della frontiera, i nodi visitati, il percorso candidato l'euristica, il costo e la priorità combinata per ogni passo dell'algoritmo:

Tabella 5.6.: ricerca del percorso tra i nodi A e L utilizzando l'algoritmo A*.

Passo	Nodo Candidato	Frontiera	Nodi Visitati	Percorso	$g(n)$	$h(n)$	$f(n) = g(n) + h(n)$
1	A	[(B, 18 + 7 = 25), (C, 21 + 7 = 28), (D, 27 + 8 = 35)]	[A]	[A]	0	23	23
2	B	[(C, 28), (D, 35), (D (via B), 20 + 8 = 28)]	[A, B]	[A, B]	18	7	25

Passo	Nodo Candidato	Frontiera	Nodi Visitati	Percorso	$g(n)$	$h(n)$	$f(n) = g(n) + h(n)$
3	D (via B)	[(C, 28), (D, 35), (E, 29)]	[A, B, D]	[A, B, D]	20	8	28
4	E	[(C, 28), (D, 35)]	[A, B, D, E]	[A, B, D, E]	29	0	29

Spiegazione dei Passaggi

1. **Passo 1**:
   - Partiamo dal nodo iniziale A.
   - Calcoliamo $f(n) = g(n) + h(n)$ per tutti i vicini:
     - $B : f(B) = 18 + 7 = 25$
     - $C : f(C) = 21 + 7 = 28$
     - $D : f(D) = 27 + 8 = 35$
   - Aggiorniamo la frontiera con i nodi B, C, D.
2. **Passo 2**:
   - Nodo B ha il valore $f(B) = 25$, il più basso nella frontiera, quindi viene esplorato.
   - Il vicino D viene aggiornato con il nuovo costo tramite B:
     - $D(viaB) : g(D) = 20, h(D) = 8, f(D) = 28.$
   - La frontiera è aggiornata.

3. **Passo 3**:

   - Nodo $D(viaB)$ ha $f(D) = 28$, il più basso nella frontiera, quindi viene esplorato.
   - Il vicino E viene aggiunto alla frontiera:
     - $E : g(E) = 29, h(E) = 0, f(E) = 29$.

4. **Passo 4**:

   - Nodo E ha $f(E) = 29$, il più basso nella frontiera, quindi viene esplorato.
   - Obiettivo raggiunto, l'algoritmo termina.

Risultato

Percorso trovato: $[A, B, D, E]$
Costo totale: 29
Euristica e costo combinati garantiscono la soluzione ottimale.

L'esempio visto mostra come l'algoritmo A* bilancia la ricerca euristica con la ricerca di costo effettivo, fornendo una soluzione ottimale se l'euristica è ammissibile e consistente. In questo caso, l'algoritmo A* suggerisce di passare per il nodo B e poi per il nodo D prima di raggiungere l'obiettivo.

5.5. Laboratorio Python

5.5.1. Esperimento 1: Algoritmi di ricerca non informata

L'algoritmo generale di ricerca visto nel paragrafo 5.2 può essere ulteriormente dettaglia in un linguaggio pseudo-Python come segue:

```
Funzione RicercaGenerale(stato_iniziale, obiettivo,
 ↪  strategia):
    # Inizializza la frontiera con lo stato iniziale
    frontiera = CreaStrutturaDati(strategia)
    Aggiungi(frontiera, stato_iniziale)

    # Inizializza l'insieme dei nodi visitati
     ↪  (opzionale, per evitare cicli)
    visitati = InsiemeVuoto()

    Mentre la frontiera non è vuota:
        # Rimuovi un nodo dalla frontiera seguendo
         ↪  la strategia scelta
        nodo_corrente = Rimuovi(frontiera)

        # Controlla se lo stato corrente soddisfa
         ↪  l'obiettivo
        Se nodo_corrente == obiettivo:
            Restituisci "Soluzione trovata!"

        # Se il nodo corrente non è stato visitato
        Se nodo_corrente non è in visitati:
            # Aggiungi il nodo corrente ai nodi
             ↪  visitati
```

```
            Aggiungi(visitati, nodo_corrente)

            # Espandi il nodo corrente per generare
             ↪  i nodi figli
            nodi_figli = GeneraFigli(nodo_corrente)

            # Aggiungi i nodi figli alla frontiera
            Per ogni nodo in nodi_figli:
                Se nodo non è in visitati e nodo non
↪  è nella frontiera:
                    Aggiungi(frontiera, nodo)

    Restituisci "Problema senza soluzione"
```

Le struttre 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 –> Usato per Greedy Best-First Search e A*, dove la priorità è determinata da funzioni di costo o euristiche.

L'implementazione in Python di questo algoritmo riportata qui di seguito discende immediatamente dalla implementazione in pseudo-python vista qui sopra. Si noti che la funzione accetta in ingresso un grafo descritto da un dizionarip, uno stato iniziale, ovvero il nome del nodo inziale nel dizionario, uno stato obiettivo, ovvero il nome del nodo da raggiungere nel dizionario, e la strategia non informata che vogliamo usare, DFS o BFS.

```
from collections import deque

def ricerca_generale(grafo, stato_iniziale,
↪  obiettivo, strategia="BFS"):
    """
    Algoritmo generale di ricerca.

    :param grafo: Dizionario che rappresenta il
↪  grafo come lista di adiacenza
    :param stato_iniziale: Nodo di partenza
    :param obiettivo: Nodo obiettivo
    :param strategia: Strategia di ricerca, "BFS"
↪  (coda) o "DFS" (stack)
    :return: Tupla contenente (percorso dal nodo
↪  iniziale al nodo obiettivo, lista dei nodi
↪  visitati) o (messaggio di fallimento, lista dei
↪  nodi visitati)
    """
    if strategia == "BFS":
        frontiera = deque([[stato_iniziale]])  #
↪  Coda per BFS
    elif strategia == "DFS":
        frontiera = [[stato_iniziale]]  # Stack per
↪  DFS
    else:
        raise ValueError("Strategia non supportata.
         ↪  Usa 'BFS' o 'DFS'.")

    visitati = set()  # Insieme dei nodi visitati
    nodi_visitati = []  # Lista per memorizzare
↪  l'ordine dei nodi visitati

    while frontiera:
```

```
        # Rimuovi un percorso dalla frontiera (FIFO
         ↪ per BFS, LIFO per DFS)
        if strategia == "BFS":
            percorso_corrente = frontiera.popleft()
        else:  # strategia == "DFS"
            percorso_corrente = frontiera.pop()

        nodo_corrente = percorso_corrente[-1]  #
↪ Ultimo nodo nel percorso

        # Controlla se abbiamo raggiunto l'obiettivo
        if nodo_corrente == obiettivo:
            return percorso_corrente, nodi_visitati
             ↪ # Restituisci il percorso completo e
             ↪ i nodi visitati

        # Se il nodo non è stato visitato
        if nodo_corrente not in visitati:
            visitati.add(nodo_corrente)  # Segna il
↪ nodo come visitato
            nodi_visitati.append(nodo_corrente)  #
↪ Aggiunge il nodo alla lista dei visitati

            # Aggiungi i vicini alla frontiera
            for vicino in grafo.get(nodo_corrente,
             ↪ []):
                if vicino not in visitati:
                    nuovo_percorso =
↪ percorso_corrente + [vicino]
                    frontiera.append(nuovo_percorso)

    return "Problema senza soluzione", nodi_visitati
```

Applichiamo la funzione di ricerca generale al seguente grafo usato nel paragrafo 5.3 che si riporta qui di seguito per comodità:

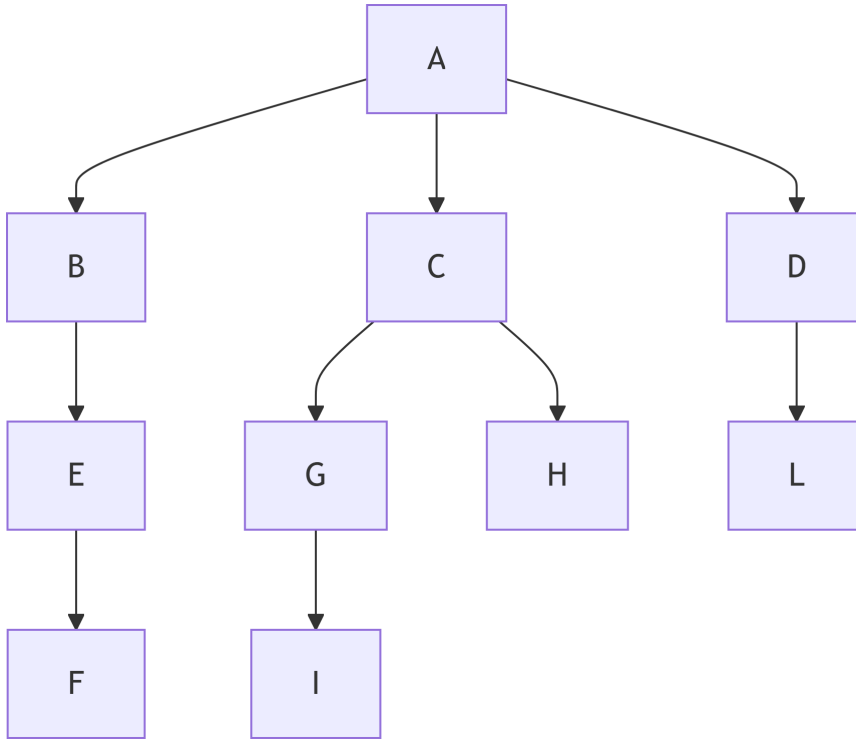


Figura 5.6.: Esempio di grafo per la funzione di ricerca generale non informata

Tradotto in Python, il grafo è rappresentato da un dizionario dove le chiavi sono i nodi e i valori sono le liste di nodi adiacenti.

```
grafo = {
    'A': ['B', 'C', 'D'],
    'B': ['E'],
```

```
    'C': ['G', 'H'],
    'D': ['L'],
    'E': ['F'],
    'G': ['I'],
    'H': [],
    'I': [],
    'F': [],
    'L': []
}
```

Creiamo una semplice funzione Python per vedere il grafo in output su schermo:

```
def stampa_grafo(grafo):
    for nodo, vicini in grafo.items():
        print(f"{nodo} -> ", end="")
        for vicino in vicini:
            print(f"{vicino}", end=" ")
        print()  # Nuova linea dopo ogni nodo
```

Proviamo a stampare per verificare di aver caricato il grafo in maniera corretta:

```
stampa_grafo(grafo)
```

```
A -> B C D
B -> E
C -> G H
D -> L
E -> F
G -> I
H ->
```

```
I ->
F ->
L ->
```

Adesso proviamo a eseguire la ricerca usando la strategia BFS:

```
# Esempi di utilizzo
print("Ricerca BFS:")
percorso, visitati = ricerca_generale(grafo, "A",
 ↪  "L", strategia="BFS")
print("Nodi visitati:", visitati)
print("Percorso completo:", percorso)
```

```
Ricerca BFS:
Nodi visitati: ['A', 'B', 'C', 'D', 'E', 'G', 'H']
Percorso completo: ['A', 'D', 'L']
```

Proviamo adesso a eseguire la ricerca usando la strategia DFS:

```
print("\nRicerca DFS:")
percorso, visitati = ricerca_generale(grafo, "A",
 ↪  "L", strategia="DFS")
print("Nodi visitati:", visitati)
print("Percorso completo:", percorso)
```

```
Ricerca DFS:
Nodi visitati: ['A', 'D']
Percorso completo: ['A', 'D', 'L']
```

Si noti come, in questo caso, la strategia DFS visiti molti più nodi rispetto alla strategia BFS per arrivare allo stesso percorso come visto nel paragrafo 5.3

5.5.2. Esperimento 2: ricerca non informata del cammino in un labirinto

Il caso del labirinto è un caso particolare di problema di ricerca in un grafo. Cominciamo con la definizione di un labirinto in formato testo. Possiamo descrivere un labirinto come una matrice di caratteri, dove i caratteri rappresentano le celle del labirinto. In particolare possiamo usare i seguenti caratteri:

- `#`: Muro
- ` `: Passaggio libero
- `S`: Inizio
- `E`: Fine
- `+`: Percorso trovato
- `-`: celle visitate

Un esempio di labirinto è il seguente:

```
###                  #########
#   ###################   # #
# ####                  # # # #
# ################### # # # #
#                     # # # #
##################### # # # #
#   ##                # # # #
# # ## ### ## ######### # # #
# #    #   ##E#         # # #
# # ## ################ # # #
### ##             #### # # #
### ############## ## # # # #
###             ##    # # # #
###### ######## ####### # # #
###### ####             #   #
S      ######################
```

Evidentemente, al di fuori della matrice, il labirinto è circondato da un muro. Prima di tutto dobbiamo leggere il labirinto da un file di testo e trasformarlo in un grafo che rappresenta il labirinto. La funzione prende in input il nome di un file che contiene un labirinto, rappresentato come una griglia 2D di caratteri. Ogni carattere rappresenta elementi diversi: '#' per i muri, 'S' per il punto di partenza, 'E' per il punto di arrivo e spazi vuoti per i percorsi percorribili. La funzione produce tre risultati: - un dizionario del grafo che mostra come le posizioni del labirinto sono collegate. Il dizionario del grafo usa coppie di coordinate (x,y) come chiavi, con ogni chiave che ha una lista delle coordinate dei vicini raggiungibili. - le coordinate della posizione iniziale - le coordinate della posizione finale.

Il funzionamento è il seguente: Per ogni posizione nella griglia del labirinto (i due cicli for uno per le righe e uno per le colonne) se non è un muro, controlla se è il punto di partenza ('S') o di arrivo ('E') e memorizza queste posizioni speciali. Poi, per ogni posizione valida, esamina le quattro possibili direzioni in cui ci si può muovere (su, destra, giù, sinistra) e aggiunge qualsiasi mossa valida alla lista delle connessioni di quella posizione nel grafo.

La principale trasformazione dei dati che avviene è la conversione da una rappresentazione a griglia 2D a una struttura a grafo dove ogni posizione è collegata ai suoi vicini accessibili.

Per esempio, se il labirinto ha un percorso dove ci si può muovere a destra e in basso dalla posizione (1,1), il grafo includerebbe qualcosa come: {(1,1): [(2,1), (1,2)]}, mostrando che dalla posizione (1,1) si possono raggiungere le posizioni (2,1) e (1,2).

```
def leggi_labirinto(file_path):
    """
    Legge un labirinto da un file ASCII e lo
 ↪  trasforma in un grafo.
```

```
    :param file_path: nome del file testo (.txt)
↪   contenente il labirinto.
    :return: Tupla contenente (labirinto, grafo,
↪   stato_iniziale, stato_finale.
    """
    with open(file_path, 'r') as file:
        labirinto = [list(line.rstrip()) for line in
↪   file]

    grafo = {}
    stato_iniziale = None
    stato_finale = None

    righe = len(labirinto)
    colonne = len(labirinto[0])

    for riga in range(righe):
        for colonna in range(colonne):
            if labirinto[riga][colonna] in (' ',
             ↪  'S', 'E'):
                nodo = (riga, colonna)
                grafo[nodo] = []

                if labirinto[riga][colonna] == 'S':
                    stato_iniziale = nodo
                elif labirinto[riga][colonna] ==
                 ↪  'E':
                    stato_finale = nodo

                # Controlla i vicini (su, giù,
                 ↪  sinistra, destra)
                for dr, dc in [(-1, 0), (1, 0), (0,
                 ↪  -1), (0, 1)]:
```

```
                    vicinino = (riga + dr, colonna +
↪ dc)
                    if 0 <= vicinino[0] < righe and
                     ↪ 0 <= vicinino[1] < colonne
                     ↪ and
                     ↪ labirinto[vicinino[0]][vicinino[1]]
                     ↪ in (' ', 'S', 'E'):
                        grafo[nodo].append(vicinino)

    return labirinto, grafo, stato_iniziale,
     ↪ stato_finale
```

Adesso ci occorre una funzione che, dato un labirinto e un percorso, stampa il labirinto con il percorso e i nodi visitati.

```
def stampa_labirinto(labirinto):
    """Stampa il labirinto in formato ASCII."""
    for riga in labirinto:
        print(''.join(riga))

def stampa_labirinto_con_percorso(labirinto,
↪ percorso, visitati):
    """Stampa il labirinto con il percorso e i nodi
     ↪ visitati."""
    labirinto_modificato = [riga.copy() for riga in
↪ labirinto]

    for riga, colonna in visitati:
        if labirinto_modificato[riga][colonna] not
         ↪ in ('S', 'E'):
            labirinto_modificato[riga][colonna] =
↪ '-'
```

```
    for riga, colonna in percorso:
        if labirinto_modificato[riga][colonna] not
         ↪  in ('S', 'E'):
            labirinto_modificato[riga][colonna] =
↪   '+'

    stampa_labirinto(labirinto_modificato)
```

Ipotizzando di aver memorizzato il labirinto nel file “labirinto.txt” possiamo leggerlo e memorizzarlo in un grafo nel seguente modo:

```
labirinto, grafo, stato_iniziale, stato_finale =
↪   leggi_labirinto("labirinto.txt")
```

Proviamo a risolvere il labirinto con l’algoritmo DFS:

```
percorso, visitati = ricerca_generale(grafo,
↪   stato_iniziale, stato_finale, strategia="DFS")

if isinstance(percorso, list):
    print("\nLabirinto con percorso trovato:")
    stampa_labirinto_con_percorso(labirinto,
↪   percorso, visitati)
else:
    print(percorso)
```

```
Labirinto con percorso trovato:
###----------------#########
#---###################---#-#
#-####---------------#-#-#-#
```

```
#-###################-#-#-#-#
#--------------------#-#-#-#
#####################-#-#-#-#
#   ##++++++++--------#-#-#-#
# # ##+### ##+#########-#-#-#
# #++++#   ##E#---------#-#-#
# #+##-#################-#-#-#
###+##-------------####-#-#-#
###+###############-##-#-#-#-#
###+++----------##----#-#-#-#
######+########-#######-#-#-#
######+####-------------#---#
S++++++#######################
```

Come possiamo vedere, l'algoritmo DFS ha trovato un percorso, ma non è l'unico possibile. Per trovare il percorso ha visitatoo molti nodi:

```
print(f"Nodi visitati: {len(visitati)}")
```

```
Nodi visitati: 193
```

Se proviamo a risolvere il labirinto con l'algoritmo BFS otteniamo il seguente risultato:

```
percorso, visitati = ricerca_generale(grafo,
 ↪  stato_iniziale, stato_finale, strategia="BFS")

if isinstance(percorso, list):
    print("\nLabirinto con percorso trovato:")
    stampa_labirinto_con_percorso(labirinto,
 ↪  percorso, visitati)
else:
    print(percorso)
```

```
Labirinto con percorso trovato:
###                 #########
#   ###################   # #
# ####                # # # #
# ################### # # # #
#                     # # # #
#################### # # # #
#---##########-       # # # #
#-#-##+###-##+######### # # #
#-#++++#---##E#         # # #
#-#+##-################ # # #
###+##---------    #### # # #
###+############## ## # # # #
###++++---------##    # # # #
######+########-#######-# # #
######+####-------------#   #
S++++++#######################
```

Per trovare il percorso ha visitatoo molti nodi:

```
print(f"Nodi visitati: {len(visitati)}")
```

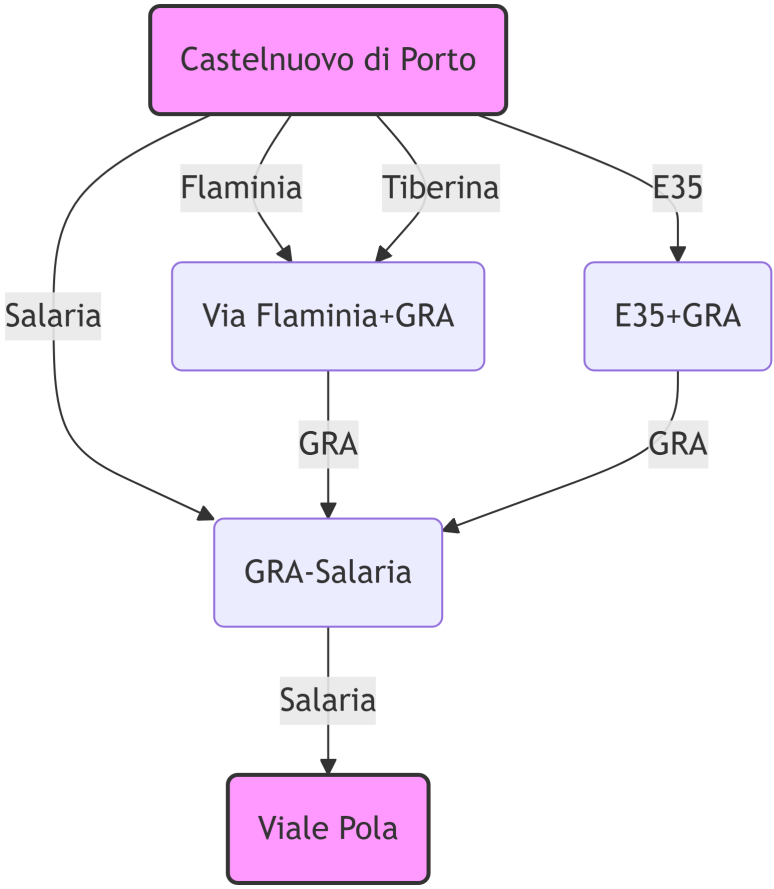
```
Nodi visitati: 76
```

Come visto nel paragrafo 5.3 anche in questo caso l'algoritmo BFS trova il percorso più breve visitando un minor numero di nodi rispetto al BFS.

5.5.3. Esperimento 3: Algoritmi di ricerca informata

Gli algoritmi di ricerca informata sono una categoria di algoritmi di ricerca che utilizzano informazioni aggiuntive oltre alla struttura

del problema per guidare la ricerca verso una soluzione. Queste informazioni aggiuntive sono spesso chiamate **euristiche**. Vediamo l'esempio proposto nel paragrafo 5.4 :



greedy best-first search

Il grafo, come visto nel paragrafo precedente, può essere rappresentato da un dizionario Python:

```
# Definizione del grafo come lista di adiacenza
grafo = {
    "A": [("B", "Flaminia"), ("C", "E35"), ("D",
     ↪  "Salaria")],
    "B": [("D", "GRA")],
    "C": [("D", "GRA")],
    "D": [("E", "Salaria")],
    "E": []
}
```

Mentre l'euristica è rappresentata da un dizionario Python con i nodi come chiavi e i valori euristici, distanza tra il nodo e l'obiettivo, come valori:

```
# Definizione dei valori euristici
euristica = {
    "A": 23,
    "B": 7,
    "C": 7,
    "D": 8,
    "E": 0
}
```

La funzione per l'implementazione dell'algoritmo **greedy best-first search** è facilmente implementabile a partire dalla funzione di ricerca generale vista nel paragrafo modificando la frontiera in una coda a priorità (min-heap) che memorizza i nodi in base al valore di euristica. Inoltre, la funzione di valutazione della frontiera viene modificata per estrarre il nodo con il valore di euristica più basso. La libreria heapq di Python può essere utilizzata per implementare una coda a priorità.

```
from heapq import heappush, heappop

def greedy_best_first_search(grafo, euristica,
↪  start, goal):
    """
    Esegue la Ricerca Greedy Best-First sul grafo
↪  fornito.

    Parametri:
        grafo (dict): Lista di adiacenza che
↪  rappresenta il grafo.
        euristica (dict): Valori euristici per ogni
↪  nodo.
        start (str): Nodo iniziale.
        goal (str): Nodo obiettivo.

    Restituisce:
        list: Percorso dal nodo iniziale al nodo
↪  obiettivo.
    """
    # Coda a priorità (min-heap) per memorizzare
     ↪  (valore euristico, nodo, percorso)
    coda_prioritaria = []
    heappush(coda_prioritaria, (euristica[start],
↪  start, [start]))

    visitati = set()  # Per tenere traccia dei nodi
↪  visitati

    while coda_prioritaria:
        # Estrai il nodo con il valore euristico più
         ↪  piccolo
        valore_h, nodo_corrente, percorso =
↪  heappop(coda_prioritaria)
```

```
        # Se l'obiettivo è raggiunto, restituisci il
         ↪ percorso
        if nodo_corrente == goal:
            return percorso

        # Se il nodo è già stato visitato, saltalo
        if nodo_corrente in visitati:
            continue

        # Segna il nodo corrente come visitato
        visitati.add(nodo_corrente)

        # Esplora i vicini
        for vicino, _ in grafo.get(nodo_corrente,
         ↪ []):
            if vicino not in visitati:
                heappush(coda_prioritaria,
↪   (euristica[vicino], vicino, percorso +
↪   [vicino]))

    # Restituisce una lista vuota se non è stato
     ↪ trovato alcun percorso
    return []
```

Proviamo a trovare il percorso più breve trale località Castelnuovo di Porto e Viale Pola, ovvero tra i nodi A e E:

```
# Esecuzione dell'algoritmo
nodo_iniziale = "A"
nodo_obiettivo = "E"
percorso = greedy_best_first_search(grafo,
↪   euristica, nodo_iniziale, nodo_obiettivo)
```

```
print("Percorso da", nodo_iniziale, "a",
 ↪  nodo_obiettivo, ":", percorso)
```

```
Percorso da A a E : ['A', 'B', 'D', 'E']
```

Quindi il percorso più breve tra Castelnuovo di Porto e Viale Pola è: [‘A’, ‘B’, ‘D’, ‘E’]

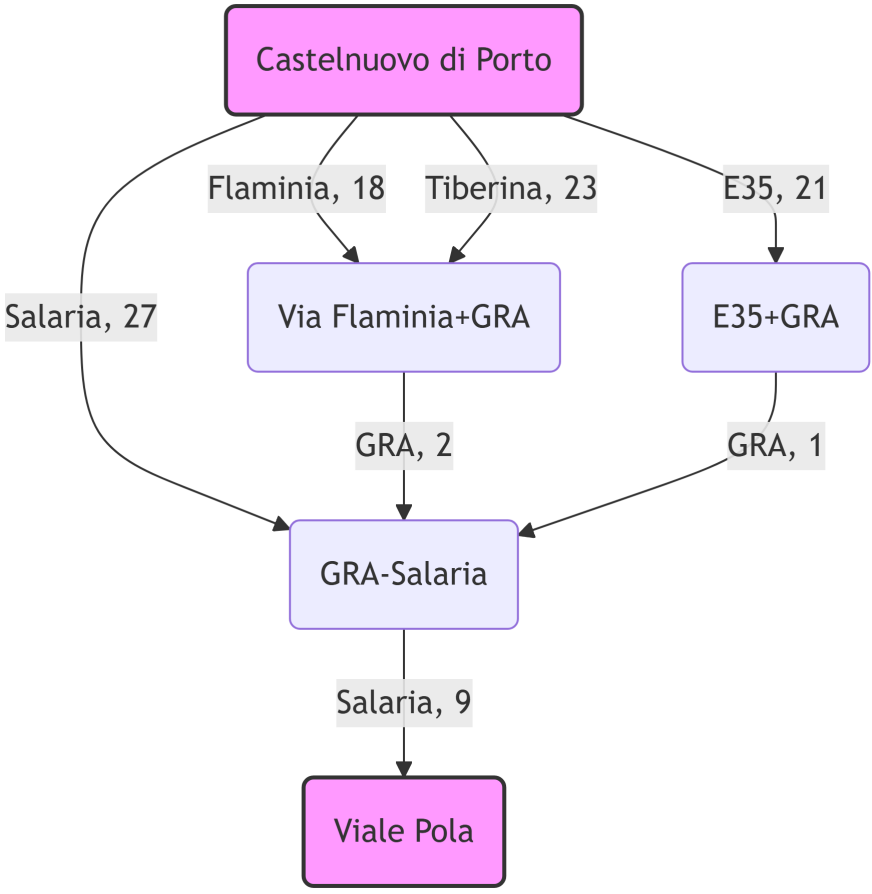
Cioè Castelnuovo di Porto -> Via Flaminia -> GRA -> Salaria -> Viale Pola.

Si invita il lettore a provare a modificare l’euristica per vedere l’effetto nella ricerca del percorso più breve.

Algoritmo A*

L’algoritmo A* è un algoritmo di ricerca informata che combina l’euristica con la ricerca di costo minimo.

Usiamo, come esempio, lo stesso grafo del paragrago Section 5.4 e l’euristica definita nel paragrafo precedente.



Come nel paragrafo precedente, il grafo può essere rappresentato da un dizionario Python. Solo che in questo caso la lista delle adiacenze, oltre ai nomi dei nodi, contiene anche il costo del percorso tra i nodi:

```
# Definizione del grafo come lista di adiacenza con
 ↪  costi
grafo = {
    "A": [("B", 18), ("C", 21), ("D", 27)],
```

```
    "B": [("D", 2)],
    "C": [("D", 1)],
    "D": [("E", 9)],
    "E": []
}
```

Mentre l'implementazione dell'euristica è la stessa del paragrafo precedente.

```
# Definizione dei valori euristici
euristica = {
    "A": 23,
    "B": 7,
    "C": 7,
    "D": 8,
    "E": 0
}
```

Una possibile implementazione dell'algoritmo A* basata sull'algoritmo generale di ricerca con l'aggiunta di una coda a priorità basata sulla somma del costo del percorso e dell'euristica, può essere la seguente:

```
from heapq import heappush, heappop

def a_star_search(grafo, euristica, start, goal):
    """
    Esegue l'algoritmo A* sul grafo fornito.

    Parametri:
        grafo (dict): Lista di adiacenza che
 ↪  rappresenta il grafo con costi.
```

```
        euristica (dict): Valori euristici per ogni
↪   nodo.
        start (str): Nodo iniziale.
        goal (str): Nodo obiettivo.

    Restituisce:
        list: Percorso dal nodo iniziale al nodo
↪   obiettivo.
    """
    # Coda a priorità (min-heap) per memorizzare
     ↪  (f(n), g(n), nodo, percorso)
    coda_prioritaria = []
    heappush(coda_prioritaria, (euristica[start], 0,
↪   start, [start]))

    visitati = set()  # Per tenere traccia dei nodi
↪   visitati

    while coda_prioritaria:
        # Estrai il nodo con il valore f(n) più
         ↪  piccolo
        _, costo_g, nodo_corrente, percorso =
↪   heappop(coda_prioritaria)

        # Se l'obiettivo è raggiunto, restituisci il
         ↪  percorso
        if nodo_corrente == goal:
            return percorso

        # Se il nodo è già stato visitato, saltalo
        if nodo_corrente in visitati:
            continue
```

```
        # Segna il nodo corrente come visitato
        visitati.add(nodo_corrente)

        # Esplora i vicini
        for vicino, costo in
         ↪ grafo.get(nodo_corrente, []):
            if vicino not in visitati:
                nuovo_costo_g = costo_g + costo
                valore_f = nuovo_costo_g +
↪   euristica[vicino]
                heappush(coda_prioritaria,
↪   (valore_f, nuovo_costo_g, vicino, percorso +
↪   [vicino]))

    # Restituisce una lista vuota se non è stato
     ↪ trovato alcun percorso
    return []
```

Proviamo a trovare il percorso più breve trale località Castelnuovo di Porto e Viale Pola, ovvero tra i nodi A e E:

```
# Esecuzione dell'algoritmo
nodo_iniziale = "A"
nodo_obiettivo = "E"
percorso = a_star_search(grafo, euristica,
 ↪ nodo_iniziale, nodo_obiettivo)

print("Percorso da", nodo_iniziale, "a",
 ↪ nodo_obiettivo, ":", percorso)
```

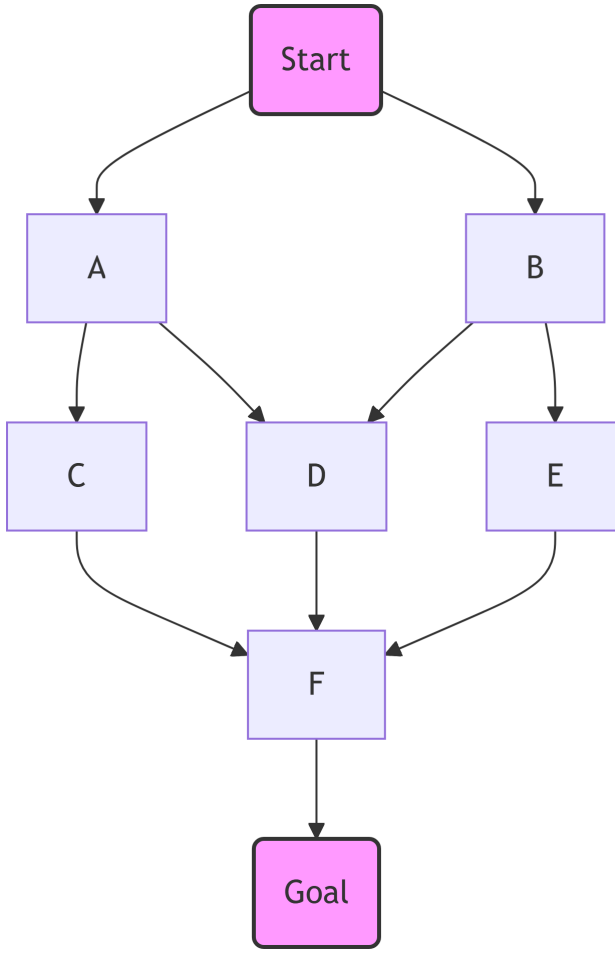
```
Percorso da A a E : ['A', 'B', 'D', 'E']
```

Quindi il percorso più breve tra Castelnuovo di Porto e Viale Pola è: [‘A’, ‘B’, ‘D’, ‘E’] Cioè Castelnuovo di Porto -> Via Flaminia -> GRA -> Salaria -> Viale Pola. Quindi l’algoritmo A* ha trovato lo stesso percorso trovato con l’algoritmo di ricerca informata greedy best first search. Questo perche la funzione di costo associato a ogni arco è esattamente l distanza stradale tra i nodi. Si invita il lettore a provare a modificare la funzione di costo per vedere l’effetto nella ricerca del percorso più breve. Ad esempio si potrebbe modificare il costo tra i nodi A e B e tra A e C impostandolo a 50 per simulare un cantiere su Flaminia e Tiberina che rallenta di molto il traffico e vedere quale percorso suggerisce A* in questo caso.

5.6. Esercizi

5.6.1. Esercizio 1: percorso più breve

Considera il seguente grafo:



1. Disegna la sequenza dei passi per trovare il percorso più breve da `S` a `G` utilizzando le seguenti strategie di ricerca:

    - Depth-First Search (DFS)
    - Breadth-First Search (BFS)

2. Quale dei due algoritmi visita meno nodi? Giustifica la tua risposta.

3. Indica quale algoritmo garantisce sempre di trovare il percorso più breve e perché.

5.6.2. Esercizio 2: progetto euristiche

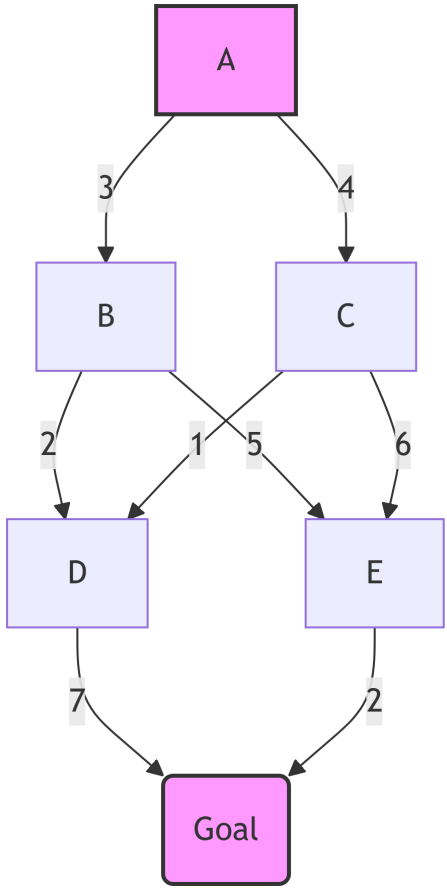
Supponi di avere una griglia 5x5 rappresentante un labirinto, con il punto di partenza `S` in `(0, 0)` e il punto di arrivo `E` in `(4, 4)`. Alcune celle sono bloccate e non possono essere attraversate:

```
S _ _ _ _
_ # _ # _
_ _ _ # _
# _ # _ _
_ _ _ _ E
```

1. Progetta due euristiche ammissibili per il problema:
    - Una basata sulla **distanza di Manhattan**.
    - Una basata sulla **distanza euclidea**.
2. Applica entrambe le euristiche per stimare il costo dal punto di partenza `(0, 0)` al punto di arrivo `(4, 4)`.
3. Quale delle due euristiche è più precisa nel guidare la ricerca?

5.6.3. Esercizio 3: algoritmi di ricerca informati

Considera il seguente grafo con costi sugli archi e valori euristici per ogni nodo:



Euristiche: - A: 10 - B: 8 - C: 6 - D: 5 - E: 3 - F: 0

1. Applica l'algoritmo Greedy Best-First Search per trovare il percorso da `A` a `F`. Registra tutti i passi in una tabella (candidato, frontiera, percorso, euristica).
2. Applica l'algoritmo A* allo stesso grafo e registra i passi.
3. Confronta i risultati ottenuti e discuti i vantaggi di A* rispetto al Greedy Best-First Search.

6. Algoritmi Equitativi

6.1. Introduzione

In questo capitolo, esploreremo gli algoritmi di ripartizione equitativa, utili per garantire che le risorse o i beni siano distribuiti in modo equo tra le parti interessate in vari scenari, dalla divisione dei beni durante un divorzio alla distribuzione di fondi di emergenza in situazioni di crisi.

I criteri di equità applicabili possono variare in base al contesto e alle esigenze specifiche. Alcuni criteri comuni esposti in (Brams and Taylor 1996) includono:

- **Proporzionalità**: Ogni partecipante riceve una quota proporzionale alle proprie pretese o contributi.
- **Invidia-zero**: Nessun partecipante deve preferire la parte assegnata a un altro partecipante alla propria parte.
- **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.
- **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 come descritto da (Robertson and Webb 1998).

6.2. glossario, regole e assunzioni

6.2.1. glossario

- **agenti partecipanti :** gli agenti o partecipanti che devono mettersi d'accordo sulla divisione di un certo numero di beni.
- **beni :** gli oggetti o le risorse che devono essere suddivisi tra gli agenti.
- **Beni divisibili :** beni che possono essere suddivisi in parti più piccole senza perdere il loro valore intrinseco (es. cibo, denaro o immobili).
- **Beni indivisibili :** beni che non possono essere suddivisi senza perdere il loro valore o funzionalità (es. un'auto, un'opera d'arte o un ruolo o incarico specifico).
- **Beni Combinati :** beni che sono una combinazione di elementi divisibili e indivisibili (es. un'azienda, un'iniziativa di beneficenza o un'organizzazione).
- **Beni con Valore Soggettivo**: come oggetti con valore sentimentale o elementi artistici o culturali, il cui valore può dipendere dal gusto personale.
- **Beni con Valore Complesso**: come proprietà immobiliari o attività commerciali, che possono avere un valore complesso che dipende dalla loro funzionalità e dalla loro posizione.
- **Beni Temporanei :** beni che possono essere utilizzati per un certo periodo di tempo e poi riassegnati (es. uso di risorse comuni, servizi o turni di lavoro).
- **Beni Digitali :** beni che possono essere duplicati e condivisi senza perdere valore (es. software, contenuti digitali).

6.2.2. Regole

Affinché la divisione di un bene S sia equa in (Moulin 2003) si osserva che devono essere soddisfatte le seguenti condizioni:

- 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.

6.2.3. Assunzioni

Gli algoritmi si basano sulle seguenti assunzionimere 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.

6.3. Algoritmi di Ripartizione Equitativa

6.3.1. Il problema della divisione equa

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. In questo caso le porzioni di torta si intendono indivisibili. Questa assunzione è importante perché se le porzioni di torta fossero divisibili, allora la soluzione sarebbe banale. 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%
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 6.2.3. Pertanto, ogni giocatore ha espresso la propria preferenza senza conoscere le preferenze degli altri partecipanti.

Un semplice algoritmo di ripartizione equa descritto in (Brams and Taylor 1996)può essere descritto come segue:

```
1. **Preparazione:**
   - Prendi le preferenze di ogni partecipante.
   - Crea una lista dei partecipanti e una lista
   delle loro preferenze per ogni porzione.
```

```
2. **Assegnazione:**
   - Per ogni porzione disponibile:
     - Trova il partecipante che preferisce di più
     quella porzione tra quelli che non hanno ancora
     ricevuto una porzione.
     - Assegna quella porzione a quel partecipante.

3. **Risultato:**
   - Crea un elenco dove ogni partecipante è
   associato alla porzione che ha ricevuto.
```

Questo algoritmo assegna le porzioni di torta in base alle preferenze dei partecipanti, garantendo che ogni partecipante riceva almeno una porzione. Applicando questo semplice algoritmo al esempio sopra descritto si ottiene la soluzione illustrata nella tabella seguente, dove è evidenziata in grassetto 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%

Si veda il paragrafo 6.4.1 per una implementazione in Python di questo algoritmo.

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:

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%

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.

6.3.2. 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? Tenendo conto che le preferenze sono le seguenti:

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

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

Alice ha il 50% di prefernenze 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 perzioni 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. Ad Alice piace il cioccolato molto più che il pistacchio. Quindi, 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.

6.3.3. Mitigazione dell'invidia

Come anticipato, è necessario approfondire il problema della divisione equa senza invidia affrontato nel dettaglio in (Procaccia 2013) al quale si rimanda per ulteriori approfondimenti sul tema. 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 indivisibili con le rispettive

valutazioni avendo ognuno a disposizione 10 punti da assegnare in totale:

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

Per tenere conto dell'invidia l'algoritmo proposto nel precedente paragrafo 6.3.1 non è sufficiente e può essere modificato come segue come discusso in (Lipton et al. 2004):

```
1. **Preparazione:**
   - Prendi le preferenze di ogni partecipante.
   - Ordina le preferenze di ogni partecipante in
   ordine decrescente.

2. **Processo Iterativo:**
   - **Prima Fase - Assegnazione Tentativa:**
     - Inizia con la porzione che ha il punteggio
     più alto per ogni partecipante.
     - Ogni partecipante "propone" la sua porzione
     preferita.
     - Se una porzione è proposta da più
     partecipanti, scegli quello che la preferisce
     di più (in caso di pari merito, scegli
     arbitrariamente).

   - **Seconda Fase - Risoluzione delle invidie:**
     - Controlla se esiste invidia:
```

```
      - Per ogni partecipante, confronta la sua
      porzione assegnata con quella di ogni altro
      partecipante. Se preferisce la porzione di un
      altro, si dice che "invidia" quell'altro
      partecipante.
    - Se c'è invidia:
      - Il partecipante invidioso "prova" a
      prendere la porzione invidiata.
      - Se il possessore attuale della porzione
      preferisce ancora la sua attuale porzione
      rispetto a quella che potrebbe ottenere,
      mantiene la sua.
      - Altrimenti, scambiano le porzioni.
    - Ripeti questo processo finché non ci sono più
    invidie.

3. **Risultato:**
   - L'allocazione finale è dove nessuna persona
   invidia l'altra.
```

Applicando questo algoritmo alle preferenze di Alice e Bruno, otteniamo la allocazione descritta dalla seguente tabella, dove 0 e 1 indicano bene non allocato e bene allocato rispettivamente:

partecipante	orologio	libro	penna	quadro
Alice	1	0	0	1
Bruno	0	1	1	0

Alice riceve l'orologio e il quadro, mentre Bruno riceve il libro e la penna, garantendo che nessuno dei due si senta invidioso dell'altro. Nel paragrafo 6.4.2 è mostrata l'implementazione di un algoritmo di divisione equa senza invidia in Python.

Per chi volesse applicare gli algoritmi equitativi in casi reali su una piattaforma online si segnala l'interessante sito web **Spliddit Algorithms** che fornisce un'ampia gamma di algoritmi equitativi (Goldman and Procaccia 2014).

6.4. Laboratorio Python

6.4.1. Esperimento 1: Algoritmo di divisione equa semplice

Il codice che segue è l'implementazione dell'algorimo di divisione equa semplice.

```
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]
}

print("Preferenze dei partecipanti:")
for p in preferenze:
    print(p, preferenze[p])


# 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}")
```

```
Preferenze dei partecipanti:
Alice [10, 50, 30, 10]
Bruno [30, 30, 10, 30]
Carla [40, 20, 20, 20]
Davide [25, 25, 25, 25]
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 algoritmpo 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.

6.4.2. Esperimento 2: Algoritmo di divisione equa senza invidia

Nwl seguito si propone una implementazione dell'algoritmo envy free in Python nel caso di beni indivisibili:

```
def allocazione_senza_invidia(beni, valutazioni):
    """
    Algoritmo senza invidia per la divisione di beni
 ↪  indivisibili tra più persone.

    beni: lista di beni da dividere
    valutazioni: dizionario con le valutazioni dei
 ↪  beni per ciascun partecipante
    """
    partecipanti = list(valutazioni.keys())

    # Inizializzazione delle assegnazioni
    assegnazione = {p: [] for p in partecipanti}
    valori_totali = {p: 0 for p in partecipanti}

    # Ordinamento dei beni in base alla somma delle
     ↪  valutazioni di tutti
    beni_ordinati = sorted(beni,
                           key=lambda x:
 ↪  sum(valutazioni[p][x] for p in partecipanti),
                           reverse=True)
```

```
    # Prima assegnazione dei beni
    for bene in beni_ordinati:
        # Trova il partecipante con il valore totale
        ↪ minimo
        min_partecipante = min(partecipanti,
↪   key=lambda p: valori_totali[p])
        assegnazione[min_partecipante].append(bene)
        valori_totali[min_partecipante] +=
↪   valutazioni[min_partecipante][bene]

    # Verifica e correzione delle invidie
    for bene in beni_ordinati:
        for p1 in partecipanti:
            for p2 in partecipanti:
                if p1 != p2 and bene in
                ↪ assegnazione[p2]:
                    if valutazioni[p1][bene] >
                    ↪ valutazioni[p2][bene] and \
                      valori_totali[p1] <
↪   valori_totali[p2]:

↪   assegnazione[p2].remove(bene)

↪   assegnazione[p1].append(bene)
                        valori_totali[p1] +=
↪   valutazioni[p1][bene]
                        valori_totali[p2] -=
↪   valutazioni[p2][bene]

    return assegnazione
```

L'algoritmo implementa una divisione equa di beni indivisibili tra più persone, cercando di minimizzare l'invidia tra i partecipanti.

Analizziamo il codice passo per passo.

1. **Definizione della Funzione**:

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

   - Definisce una funzione chiamata `allocazione_senza_invidia` che accetta due parametri:
     - beni: lista degli oggetti da dividere
     - valutazioni: dizionario con le valutazioni di ogni partecipante per ogni bene

2. **Inizializzazione delle Assegnazioni**:

```
partecipanti = list(valutazioni.keys())
assegnazione = {p: [] for p in partecipanti}
valori_totali = {p: 0 for p in partecipanti}
```

   Questa fase

   - Estrae la lista dei partecipanti
   - Crea un dizionario vuoto per tracciare i beni assegnati
   - Inizializza a zero i valori totali per ogni partecipante

3. **Ordinamento dei Beni**:

```
beni_ordinati = sorted(beni,
                       key=lambda x:
 ↪  sum(valutazioni[p][x] for p in
 ↪  partecipanti), reverse=True)
```

   - I beni vengono ordinati in base al loro valore totale (somma delle valutazioni di tutti i partecipanti) in ordine decrescente.

4. **Prima assegnazione dei Beni**:

```
for bene in beni_ordinati:
  min_partecipante = min(partecipanti,
 ↪  key=lambda p: valori_totali[p])
  assegnazione[min_partecipante].append(bene)
  valori_totali[min_partecipante] +=
 ↪  valutazioni[min_partecipante][bene]
```

Per ogni bene:

- Trova il partecipante con il minor valore totale
- Assegna il bene a quel partecipante
- Aggiorna il valore totale del partecipante

5. **Verifica e Correzione delle Invidie**:

```
for bene in beni_ordinati:
for p1 in partecipanti:
    for p2 in partecipanti:
        if p1 != p2 and bene in
         ↪  assegnazione[p2]:
            if valutazioni[p1][bene] >
             ↪  valutazioni[p2][bene] and \
               valori_totali[p1] <
 ↪  valori_totali[p2]:
                assegnazione[p2].remove(bene)
                assegnazione[p1].append(bene)
                valori_totali[p1] +=
 ↪  valutazioni[p1][bene]
                valori_totali[p2] -=
 ↪  valutazioni[p2][bene]
```

Questa fase verifica e corregge eventuali invidie:

- Controlla ogni coppia di partecipanti

- Se un partecipante valuta più un bene assegnato a un altro e ha un valore totale minore di quello dell'altro, effettua uno scambio di beni.

6. **Ritorno delle Assegnazioni**:

```
return assegnazione
```

- Restituisce il dizionario delle assegnazioni finali.

Caso d'Uso

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', 'quadro']
valutazioni = {
    'Alice': {'orologio': 4, 'libro': 2, 'penna': 2,
    ↪  'quadro': 2},
    'Bruno': {'orologio': 2, 'libro': 3, 'penna': 3,
    ↪  'quadro': 2}
}
```

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

```
risultato = allocazione_senza_invidia(beni,
 ↪  valutazioni)
print("Allocazione finale:")
for persona, oggetti in risultato.items():
    print(f"{persona}: {oggetti}")
```

```
Allocazione finale:
Alice: ['orologio', 'quadro']
Bruno: ['libro', 'penna']
```

6.5. esercizi

6.5.1. Esercizio 1: divisione equa di una torta

Hai quattro partecipanti (Alice, Bruno, Carla, e Davide) che devono dividersi una torta con quattro porzioni, ciascuna con percentuali di preferenza diverse:

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%

1. Assegna le porzioni usando l'algoritmo semplice di ripartizione equa descritto nel testo.
2. Analizza la distribuzione ottenuta e discuti se i partecipanti potrebbero sentirsi invidiosi delle porzioni assegnate.
3. Proponi una soluzione alternativa per mitigare eventuali invidie.

6.5.2. Esercizio 2: divisione beni indivisibili con valori soggettivi

Un gruppo di tre amici deve dividersi un insieme di beni con valori soggettivi assegnati a ciascun bene da ogni partecipante. I beni e le rispettive valutazioni sono:

Bene	Valore per Alice	Valore per Bruno	Valore per Carla
Laptop	€900	€850	€1000
Smartphone	€700	€750	€700
Tablet	€400	€300	€350

1. Calcola il valore totale dei beni assegnati a ciascun partecipante se si utilizza un algoritmo semplice basato su massimizzazione delle preferenze individuali.
2. Valuta se l'assegnazione è equa e giustifica la tua risposta.
3. Proponi un'assegnazione alternativa che riduca l'invidia tra i partecipanti.

6.5.3. Esercizio 3: divisione di beni indivisibili con valutazioni soggettive

Due amici, Alice e Bruno, devono dividersi i seguenti beni indivisibili, con un massimo di 10 punti da distribuire in totale come valutazione personale per ogni partecipante:

Bene	Valutazione di Alice	Valutazione di Bruno
Bicicletta	6	4
Orologio	2	3
Libro	2	3

1. Utilizza l'algoritmo senza invidia descritto nel testo per assegnare i beni.
2. Spiega perché la soluzione ottenuta è senza invidia, oppure discuti eventuali problemi riscontrati.
3. Calcola l'equità in termini di punti assegnati e confronta con un'assegnazione casuale dei beni.

7. Algoritmi Predittivi

7.1. Introduzione

Gli algoritmi predittivi rappresentano un salto qualitativo rispetto agli algoritmi deterministici e probabilistici visti nei capitoli precedenti(Bishop 2006). Mentre gli algoritmi deterministici seguono regole ben definite per raggiungere una soluzione e gli algoritmi probabilistici utilizzano la probabilità per gestire l'incertezza, gli algoritmi predittivi apprendono dai dati per costruire modelli capaci di fare previsioni su eventi futuri.

Differenze principali:

1. **Deterministici**:
    - Producono lo stesso risultato ogni volta con gli stessi input.
    - Non gestiscono incertezza o variabilità nei dati.
    - Esempio: algoritmi di ricerca in grafi (DFS, BFS).
2. **Probabilistici**:
    - Includono l'incertezza nei calcoli.
    - Basati su modelli matematici che stimano la probabilità di eventi.
    - Esempio: filtri email o algoritmi Bayesiani.
3. **Predittivi**:

- Si basano sull'apprendimento dai dati per costruire modelli.
- Adattano il comportamento in base a nuove informazioni.
- Possono gestire sia dati complessi che non lineari.
- Esempi: regressione lineare, reti neurali, support vector machine (SVM).

La vera forza degli algoritmi predittivi risiede nella loro capacità di migliorare le decisioni grazie all'apprendimento automatico. Possono affrontare scenari complessi dove non esistono regole fisse e dove i dati contengono rumore o incertezza. Gli algoritmi di predizione sono usati per creare modelli basati sull' apprendedimento 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 (Murphy 2012):

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 neurali[1].
2. **Apprendimento Non Supervisionato**: Questi algoritmi scoprono pattern nascosti o strutture nei dati non etichettati. Tecniche comuni sono la clusterizzazione e la riduzione della dimensionalità[1].
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 ambiente[1].

Classificazione Basata sul Tipo di Output:

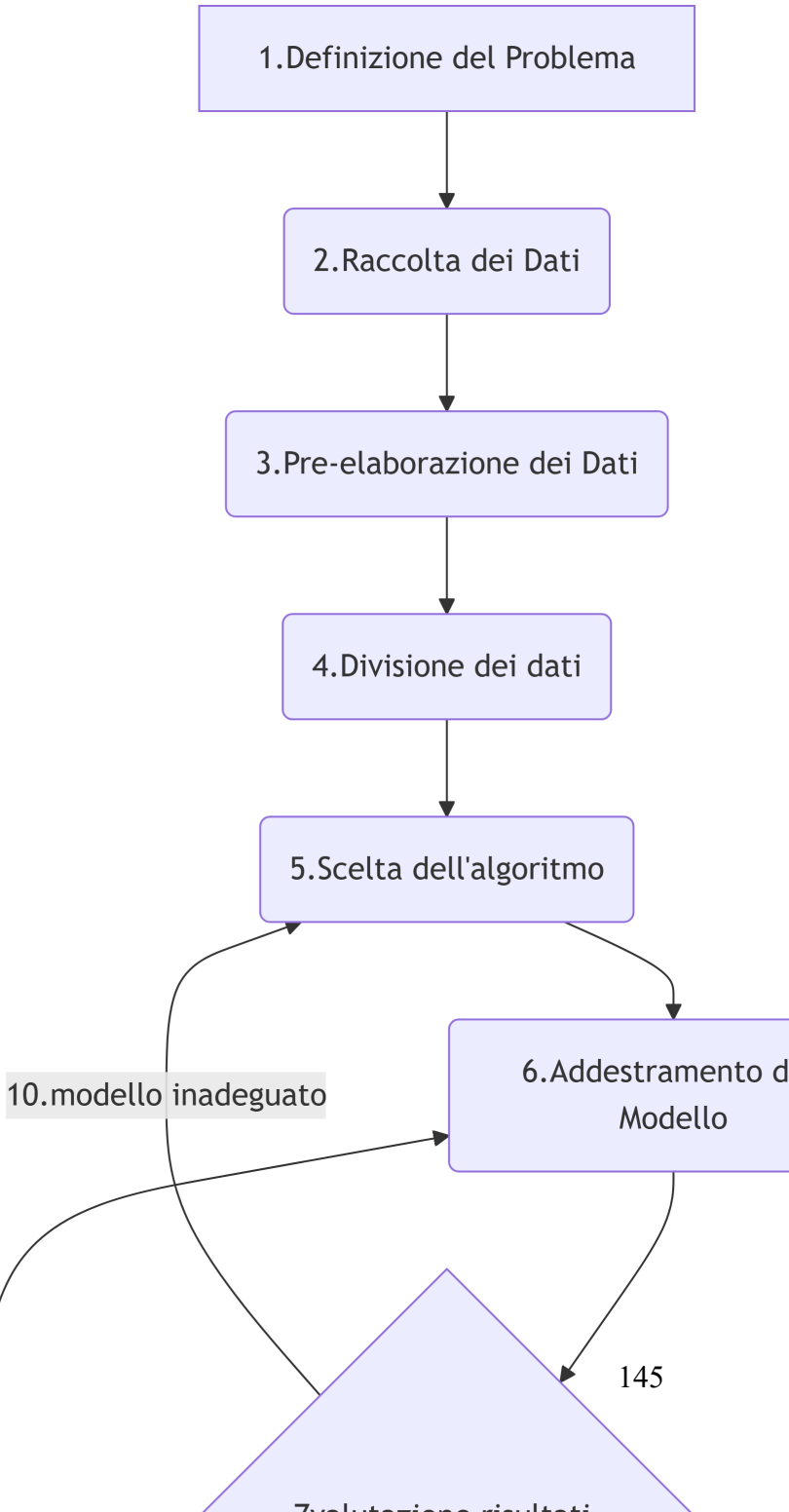
1. **Classificazione**: Quando l'output è una categoria, come "spam" o "non spam" in un filtro di posta elettronica, si parla di classificazione. Gli algoritmi di classificazione assegnano un'etichetta discreta a un'istanza di input[1].
2. **Regressione**: Se l'output è un valore continuo, come il prezzo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

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 overfitting[2].
2. **Random Forest**: Una collezione di alberi decisionali che riduce il rischio di overfitting e gestisce meglio le variabili non correlate[2].
3. **Support Vector Machine (SVM)**: Trovano il miglior iperpiano che separa i dati in classi. Sono efficaci in spazi ad alta dimensionalità[2].
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 costosi[2].
5. **Reti Neurali**: Sono modelli ispirati al funzionamento del cervello umano e possono catturare relazioni complesse nei dati[1].

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 grandi[2]. 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
2.Raccolta dei Dati
3.Pre-elaborazione dei Dati
4.Divisione dei dati
5.Scelta dell'algoritmo
6.Addestramento d
Modello
10.modello inadeguato
145

Nei prossimi paragrafi si presenterà una descrizione di ognuno di questi passi con la descrizione di un caso concreto di previsione della probabilità che un imputato in libertà vigilata commetta un nuovo reato. Si veda (Inc. 2018) per la descrizione di un applicativo di questo tipo in uso presso gli organi giudiziari negli Stati Uniti.

7.2. Definizione del Problema

Questp 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**: Prevedere la probabilità che un imputato in libertà vigilata commetta un nuovo reato.

- **Obiettivo**: Stimare il rischio di recidiva.
- **Dati necessari**: Età, tipo di reato precedente, durata della libertà vigilata.
- **Ipotesi**: Alcuni fattori, come precedenti specifici, possono aumentare il rischio.

7.3. 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**: Raccolta di dati da archivi giudiziari, registri penitenziari e rapporti sociali.

- **Fonti**: Sentenze, relazioni degli assistenti sociali.
- **Dati**:
  - Età: 25, 30, 35, 40, 45, 50, 55, 60, 65, 70
  - Tipo di reato: furto, furto con violenza, omicidio, omicidio con violenza, …
  - Durata della libertà vigilata: 1 mese, 3 mesi, 6 mesi, 1 anno, 2 anni, 3 anni, 4 anni, 5 anni, 6 anni, 7 anni
  - …
- **Risultato**:
  - Rischio di recidiva: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
- **Considerazioni**: Garantire la protezione dei dati personali e una raccolta dei dei dati bilanciata tra le classi sottostanti il fenomeno da studiare per evitare una eccessiva polarizzazione dei dati.

7.4. 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**:

- **Pulizia**: Rimuovere duplicati o incongruenze nei dati delle sentenze.

- **Gestione dei valori mancanti**: Stimare dati mancanti, come la durata del procedimento.

- **Normalizzazione**: Uniformare i dati, ad esempio convertendo valute in una stessa unità.

7.5. 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**:

- **Divisione**: 70% addestramento, 15% validazione, 15% test.

7.6. Scelta dell' algoritmo

La scelta dell'algoritmo determina l'approccio con cui il modello analizzerà i dati e farà previsioni (James et al. 2013). 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**: In questo si potrebbe usare un algoritmo di regressione, come una Random Forest Regressor, per stimare la probabilità di recidiva.

7.7. 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 (Hastie, Tibshirani, and Friedman 2009). 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.

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. Nel processo di addestramento occorre spesso intervenire manualmente per regolare i parametri del modello, come la dimensione del batch, la velocità di apprendimento e la regolarizzazione per evitare fenomeni come 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.

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. **Esempio Pratico**: Nel caso in studio si tratterà di addestrare il modello scelto con dati storici su imputati e loro comportamenti post-processo.

7.8. 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. Le metriche di valutazione variano a seconda del tipo di problema:

- **problemi di classificazione**, si utilizzano metriche come: **Matrice di confusione**: una tabella che mostra il numero di casi classificati correttamente e erroneamente in ogni classe.
  - **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.
- **problemi di regressione**, 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.

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**: nel caso in studio di modello di machine learning addestrato per predire la

probabilità di recidività di un reo in base ai dati raccolti sullo stesso. 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.

7.9. 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: Nel caso in studio si procederà a regolare i parametri della Random Forest Regressor, come il numero di alberi (n_estimators) o la profondità massima degli alberi (max_depth), per migliorare le prestazioni. Una possibile metodologia di ottimizzazione potrebbe essere di utilizzare una ricerca a griglia per identificare i migliori iperparametri. Ad esempio, testare diversi valori di n_estimators (100, 200, 500) e max_depth (5, 10, 15). La validazione incrociata può essere utilizzata per valutare le prestazioni del modello su diversi set di dati di addestramento e convalida.

7.10. 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: nel caso in studio si procederà a creare un'applicazione web che consenta agli utenti di inserire i dati di input e ricevere le previsioni del modello. Questa applicazione sarà ospitata su un server web e sarà accessibile tramite un'interfaccia web nella quale sarà necessario implementare un sistema di autenticazione per garantire che solo gli utenti autorizzati possano accedere al modello. Se si prevede che il modello sarà utilizzato in un'applicazione mobile, sarà necessario sviluppare un'interfaccia utente per dispositivi mobili che consenta agli utenti di inserire i dati di input e ricevere le previsioni. 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 previsioni.

7.11. 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(Kuhn and Johnson 2013). 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**: Nel caso in studio, il monitoraggio del modello di previsione potrebbero includere la verifica regolare dell'MSE e del'R^2 su nuovi dati. Mentre la manutenzione potrebbe includere l'aggiornamento periodico, ad esempio ogni 6 mesi, del modello con nuovi dati per garantire prestazioni ottimali.

7.12. Laboratorio Python

7.12.1. Esperimento 1: Predizione della recidiva parte A: creazione di un dataset di addestramento e test simulato

Vediamo un esempio in Python che implementa il processo predittivo per il caso giuridico di previsione della probabilità di recidiva utilizzando una **Random Forest Regressor**. Questo esempio utilizza diviso in 3 parti si basa su dati simulati per illustrare il flusso completo, dalla generazione dei dati fino alla valutazione del modello.

> **Caution**
>
> Se non è stato già fatto, installare le librerie necessarie con il seguente comando da eseguire nella shell:
>
> ```
> pip install numpy pandas sklearn matplotlib
> ```

In questa primo esperimento l'obiettivo principale è generare dati dall'aspetto realistico su persone che sono in libertà vigilata (rilascio supervisionato dopo un reato) e calcolare il loro rischio di recidiva. Questi dati simulati verranno successivamente utilizzati per addestrare e testare un modello predittivo, che è una pratica comune nel machine learning quando si vuole sperimentare prima di utilizzare dati reali e sensibili. Il codice non riceve input esterni dagli utenti. Invece, genera i propri dati utilizzando la generazione di numeri casuali. Tuttavia, utilizza diversi parametri predefiniti per controllare la simulazione:

- Numero di campioni (1000 persone)
- Fasce d'età (da 18 a 70 anni)
- Durata della libertà vigilata (da 1 a 60 mesi)
- Tipi di reati precedenti (furto, reato violento, omicidio)
- Stato lavorativo (lavoro stabile o no)
- Supporto familiare (presente o assente)

Il codice segue una sequenza chiara: genera caratteristiche individuali casualmente, poi le combina usando una formula ponderata per calcolare il rischio. Un passaggio cruciale è il "clipping" del punteggio di rischio finale per assicurarsi che rimanga tra 0 e 1, poiché le probabilità non possono essere negative o maggiori del 100%.

Il seed casuale (impostato a 42) garantisce che ogni volta che esegui questo codice, ottieni esattamente gli stessi dati "casuali", il che è importante per esperimenti scientifici riproducibili. Infine, tutti i dati

generati vengono organizzati in un DataFrame pandas (pensalo come un foglio di calcolo digitale) e visualizza le prime righe così puoi vedere com'è fatto il dataset simulato.

Questo approccio permette a ricercatori e sviluppatori di testare i loro algoritmi di predizione su dati realistici senza problemi di privacy, prima di applicarli a veri fascicoli giudiziari.

```
import numpy as np
import pandas as pd

# Simulazione dei dati con relazioni realistiche
np.random.seed(42)
num_samples = 1000

# Generazione delle variabili
eta = np.random.randint(18, 70, size=num_samples)
durata_liberta_vigilata = np.random.randint(1, 60,
 ↪  size=num_samples)  # Durata in mesi
reato_precedente = np.random.choice([0, 1, 2],
 ↪  size=num_samples, p=[0.5, 0.3, 0.2])  # Furto,
 ↪  violenza privata, omicidio
lavoro_stabile = np.random.choice([0, 1],
 ↪  size=num_samples, p=[0.6, 0.4])
supporto_familiare = np.random.choice([0, 1],
 ↪  size=num_samples, p=[0.4, 0.6])

# Calcolo del rischio di recidiva basato su regole
rischio_recidiva = (
    0.6 * (1 - lavoro_stabile) +
    0.5 * (1 - supporto_familiare) +
    0.4 * (reato_precedente / 2) +
    0.3 * (1 / durata_liberta_vigilata) +
    0.2 * (1 / (eta - 17))
```

```
)
rischio_recidiva = np.clip(rischio_recidiva, 0, 1)
 ↪  # Limitare il rischio tra 0 e 1

# Creazione del DataFrame
data = {
    'eta': eta,
    'durata_liberta_vigilata':
     ↪  durata_liberta_vigilata,
    'reato_precedente': reato_precedente,
    'lavoro_stabile': lavoro_stabile,
    'supporto_familiare': supporto_familiare,
    'rischio_recidiva': rischio_recidiva
}

df = pd.DataFrame(data)

# Visualizzazione iniziale dei dati
print("Esempio di dati simulati:")
print(df.head())
```

```
Esempio di dati simulati:
   eta  durata_liberta_vigilata  reato_precedente
   lavoro_stabile  \
0   56                       35                 2
0
1   69                       51                 1
0
2   46                       15                 0
1
3   32                       25                 1
0
```

```
4   60                          55                  1
1

   supporto_familiare  rischio_recidiva
0                   0          1.000000
1                   0          1.000000
2                   0          0.526897
3                   0          1.000000
4                   0          0.710106
```

Si noti che i dati generati sono memorizzati in una struttura dati chiamata Dataframe. Si tratta di una particolare struttura dati fornita dalla libreria pandas di Python che rappresenta essenzialmente una tabella, molto simile a un foglio di calcolo di Excel o a una tabella di un database. Possiamo immaginarla come una griglia con righe e colonne: ogni riga rappresenta un record (in questo caso, una persona) e ogni colonna rappresenta una caratteristica o variabile (come età, tipo di reato, ecc.). Questa struttura dati è diventata molto popolare nel mondo del machine learning perché è molto flessibile e può gestire una grande quantità di dati in modo efficiente. Con questa struttura dati, è possibile eseguire operazioni come la selezione di righe o colonne specifiche, la manipolazione dei dati, la creazione di nuovi dati basati su quelli esistenti e molto altro. Inoltre, le librerie di machine learning come scikit-learn richiedono spesso che i dati siano organizzati in questo formato per poter essere utilizzati.

7.12.2. Esperimento 2: Predizione della recidiva parte B: Addestramento e test del modello

In questo esperimento, il modello Random Forest Regressor viene addestrato utilizzando i dati simulati e quindi viene utilizzato per

fare previsioni sui dati di test. Per prima cosa, i dati sono divisi in input (X) e output (y), dove X contiene tutte le caratteristiche (ad esempio, età, tipo di reato precedente, ecc.) e y contiene il rischio di recidiva. Successivamente, i dati sono divisi in set di addestramento e test utilizzando la funzione train_test_split di scikit-learn. Il modello Random Forest Regressor viene addestrato utilizzando i dati di addestramento e quindi viene utilizzato per fare previsioni sui dati di test.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,
 ↪  r2_score

# Divisione dei dati in input (X) e output (y)
X = df.drop('rischio_recidiva', axis=1)
y = df['rischio_recidiva']

# Divisione in training e test set
X_train, X_test, y_train, y_test =
 ↪  train_test_split(X, y, test_size=0.2,
 ↪  random_state=42)

# Creazione e addestramento del modello Random
 ↪  Forest Regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

# Predizione sui dati di test
y_pred = rf_model.predict(X_test)

# Valutazione del modello
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print("\nRisultati del modello:")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R^2 Score: {r2:.4f}")
```

```
Risultati del modello:
Mean Squared Error (MSE): 0.0004
R^2 Score: 0.9966
```

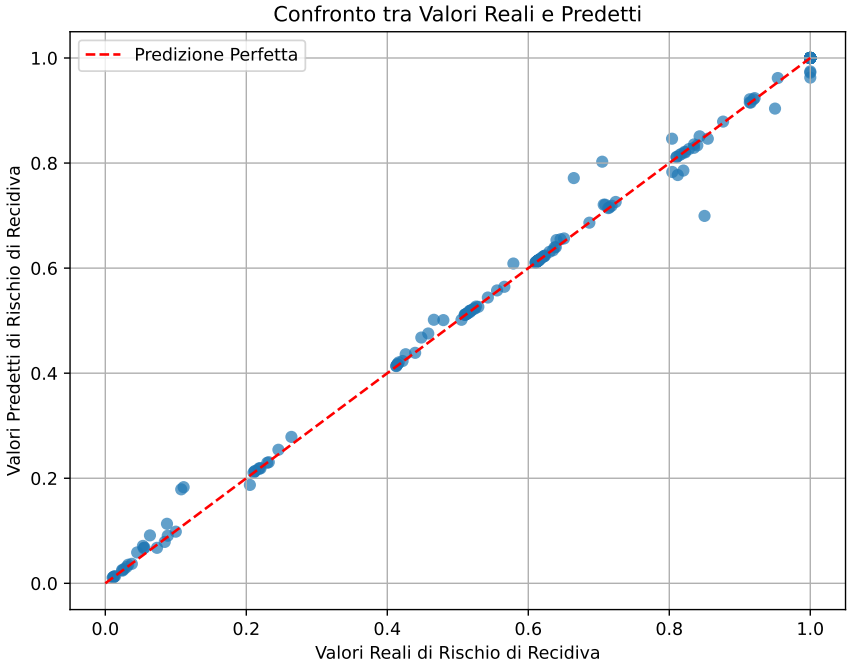
7.12.3. Esperimento 3: Predizione della recidiva parte C: Ottimizzazione degli iperparametri del modello e visualizzazione dei risultati

In questa parte dell'esperimento, viene visualizzato un grafico per confrontare i valori reali di rischio di recidiva con i valori predetti dal modello. Inoltre, viene utilizzato GridSearchCV per ottimizzare gli iperparametri del modello Random Forest Regressor. Questo processo di ottimizzazione è importante perché aiuta a trovare la combinazione migliore di parametri che migliorano le prestazioni del modello. Infine, i risultati dell'ottimizzazione vengono visualizzati in un grafico.

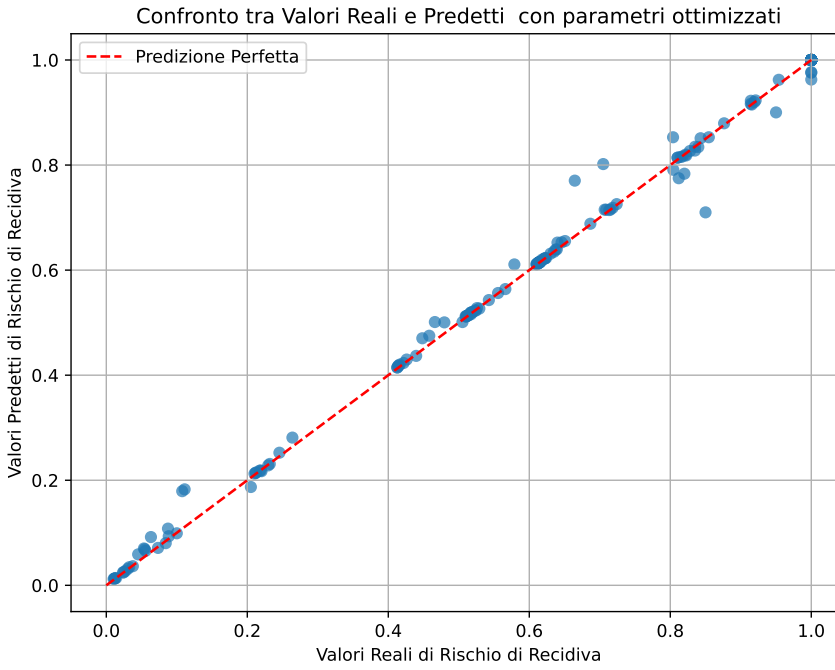
```
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV

# Visualizzazione dei risultati
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.plot([0, 1], [0, 1], '--', color='red',
 ↪  label='Predizione Perfetta')
```

```
plt.xlabel("Valori Reali di Rischio di Recidiva")
plt.ylabel("Valori Predetti di Rischio di Recidiva")
plt.title("Confronto tra Valori Reali e Predetti")
plt.legend()
plt.grid()
plt.show()

# Ottimizzazione degli iperparametri con
 ↪ GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(estimator=rf_model,
 ↪ param_grid=param_grid, cv=5,
 ↪ scoring='neg_mean_squared_error', verbose=1,
 ↪ n_jobs=-1)
grid_search.fit(X_train, y_train)

print("\nMigliori parametri trovati con
 ↪ GridSearchCV:")
print(grid_search.best_params_)

# Addestramento del modello ottimizzato
best_rf_model = grid_search.best_estimator_

# Predizione e valutazione con il modello
 ↪ ottimizzato
y_pred_optimized = best_rf_model.predict(X_test)
optimized_mse = mean_squared_error(y_test,
 ↪ y_pred_optimized)
```

```
optimized_r2 = r2_score(y_test, y_pred_optimized)

# Visualizzazione dei risultati
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_optimized, alpha=0.7)
plt.plot([0, 1], [0, 1], '--', color='red',
 ↪  label='Predizione Perfetta')
plt.xlabel("Valori Reali di Rischio di Recidiva")
plt.ylabel("Valori Predetti di Rischio di Recidiva")
plt.title("Confronto tra Valori Reali e Predetti
 ↪  con parametri ottimizzati")
plt.legend()
plt.grid()
plt.show()

print("\nRisultati del modello ottimizzato:")
print(f"Mean Squared Error (MSE):
 ↪  {optimized_mse:.4f}")
print(f"R^2 Score: {optimized_r2:.4f}")
```




```
Fitting 5 folds for each of 27 candidates, totalling
135 fits

Migliori parametri trovati con GridSearchCV:
{'max_depth': 10, 'min_samples_split': 2,
'n_estimators': 300}
```




```
Risultati del modello ottimizzato:
Mean Squared Error (MSE): 0.0003
R^2 Score: 0.9967
```

7.13. Esercizi

7.13.1. Esercizio 1: Analisi delle variabili significative

Obiettivo: Identificare le variabili più importanti nel determinare il rischio di recidiva.

Supponendo di avere un modello di **Random Forest Regressor** già addestrato con i dati di rischio di recidiva:

1. Utilizza la funzione `feature_importances_` della Random Forest per identificare l'importanza relativa delle variabili di input.
2. Ordina le variabili in base alla loro importanza decrescente.
3. Rispondi alle seguenti domande:

    - Quali sono le tre variabili più significative?
    - Come pensi che queste variabili influenzino il rischio di recidiva?

7.13.2. Esercizio 2: Creazione di un dataset realistico

Obiettivo: Simulare un dataset che rifletta il comportamento reale di imputati in libertà vigilata.

1. Simula un dataset di almeno 500 imputati includendo le seguenti variabili:

    - Età.
    - Tipo di reato precedente (furto, violenza, omicidio).
    - Durata della libertà vigilata (in mesi).
    - Presenza di un lavoro stabile (sì/no).
    - Supporto familiare (sì/no).

2. Assegna il rischio di recidiva utilizzando una funzione che tenga conto delle variabili sopra indicate (ad esempio, un punteggio più alto per imputati senza lavoro stabile o con reati più gravi).
3. Visualizza la distribuzione del rischio di recidiva nel dataset utilizzando un grafico.

7.13.3. Esercizio 3: Valutazione di un modello predittivo

Obiettivo: Valutare le prestazioni di un modello predittivo per il rischio di recidiva.

1. Addestra un modello di **Random Forest Regressor** utilizzando un dataset simulato.
2. Dividi i dati in training e test set con una proporzione 80/20.
3. Valuta il modello utilizzando le seguenti metriche:
    - **Mean Squared Error (MSE)**.
    - **R^2 Score**.
4. Interpreta i risultati:
    - Il modello è efficace? Quali sono i possibili miglioramenti?
    - Quali interventi (ad esempio, ottimizzazione degli iperparametri) potresti fare per migliorare le prestazioni?

Parte II.

Machine learning

8. Apprendimento Supervisionato

8.1. Introduzione

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).

8.2. 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). Tra le tecniche utilizzate si possono citare approcci come le **reti neurali** e le **macchine a vettori di supporto (SVM)**
- **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 l'applicazione multipla di **macchine a vettori di supporto (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.

8.3. 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.

8.4. 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.

8.4.1. 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.

8.4.2. 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).

8.4.3. 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.

8.4.4. 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.

8.4.5. Support Vector Machines (SVM)

Le Support Vector Machines (SVM) sono algoritmi molto potenti sia per la classificazione. 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.

8.4.6. k-Nearest Neighbors (k-NN)

Il k-Nearest Neighbors (k-NN) è un algoritmo di classificazione 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.

8.4.7. 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, ma richiedono un'attenta configurazione dei parametri e una grande quantità di dati per addestramento.

- **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.

8.4.8. 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.

8.4.9. 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.

8.4.10. 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.

8.4.11. 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.

8.5. Apprendimento per Rinforzo

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. 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.

8.5.1. Algoritmi Principali

- **Algoritmo di Monte Carlo (MC)**: Questo algoritmo valuta le politiche basandosi sui risultati di una serie di episodi completi. È un algoritmo **on-policy**, il che significa che l'agente deve seguire la politica corrente per apprendere.
- **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.

8.5.2. 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.

8.6. 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.

8.7. Valutazione dei Modelli

La **valutazione dei modelli** è un passo critico per garantire che un modello di apprendimento supervisionato sia accurato, 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. Innanzitutto, è importante sottolineare che la valutazione dei modelli si effettua sui **dati di test**, non sui dati di addestramento. Il modello deve dimostrare la capacità di **generalizzare**, ossia di fare previsioni accurate su nuovi dati che non ha mai visto prima.

8.7.1. Valutazione nei Problemi di Classificazione

Per i problemi di classificazione binaria (due classi, ad esempio A e B), le metriche di valutazione più comuni si basano sulla misurazione dei seguenti valori:

- **Veri Positivi (VP)**: il modello ha correttamente predetto che un certo numero di casi appartiene alla classe A e questi effettivamente appartengono alla classe A.
- **Falsi Positivi (FP)**: il modello ha predetto che un certo numero di casi appartiene alla classe A, ma in realtà appartengono alla classe B.
- **Veri Negativi (VN)**: il modello ha correttamente predetto che un certo numero di casi appartiene alla classe B e questi effettivamente appartengono alla classe B.

- **Falsi Negativi (FN)**: il modello ha predetto che un certo numero di casi appartiene alla classe B, ma in realtà appartengono alla classe A.

Le metriche di valutazione comuni includono:

1. **Matrice di Confusione**
   La matrice di confusione riassume i valori di VP, FP, VN e FN in una tabella. È utile per analizzare in dettaglio le prestazioni del modello. La forma standard della matrice di confusione per un problema di classificazione binaria è la seguente:

$$\text{Matrice di Confusione} = \begin{bmatrix} VP & FP \\ FN & VN \end{bmatrix}$$

   Questa rappresentazione permette di identificare rapidamente dove il modello ha successo e dove commette errori.

2. **Accuratezza**
   È la proporzione di previsioni corrette sul totale delle previsioni. Tuttavia, in presenza di classi sbilanciate, l'accuratezza può essere ingannevole.

$$\text{Accuratezza} = \frac{VP + VN}{VP + FP + VN + FN};$$

3. **Precisione e Recall**

   - **Precisione**: misura la proporzione di veri positivi rispetto al totale delle predizioni positive.

$$\text{Precisione} = \frac{VP}{VP + FP};$$

- **Recall (o Sensibilità)**: misura la proporzione di veri positivi rispetto al totale dei casi positivi reali.

$$\text{Recall} = \frac{VP}{VP + FN};$$

4. **F1-Score**
   È la media armonica tra precisione e recall, utile quando è necessario bilanciare entrambe le metriche.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precisione} \cdot \text{Recall}}{\text{Precisione} + \text{Recall}};$$

5. **AUC-ROC (Area Under the Curve - Receiver Operating Characteristic)**
   La curva **ROC** (*Receiver Operating Characteristic*) è un grafico che mostra la capacità di un classificatore binario di distinguere tra due classi, variando la soglia di classificazione. Si costruisce tracciando il **tasso di veri positivi** TPR contro il **tasso di falsi positivi** FPR per diverse soglie di decisione. L'AUC è l'area sotto questa curva:

- **AUC = 1**: Modello perfetto.
- **AUC = 0.5**: Modello casuale (nessuna capacità di discriminazione).
- **AUC < 0.5**: Modello peggiore del caso casuale (probabile errore nel modello o nei dati).

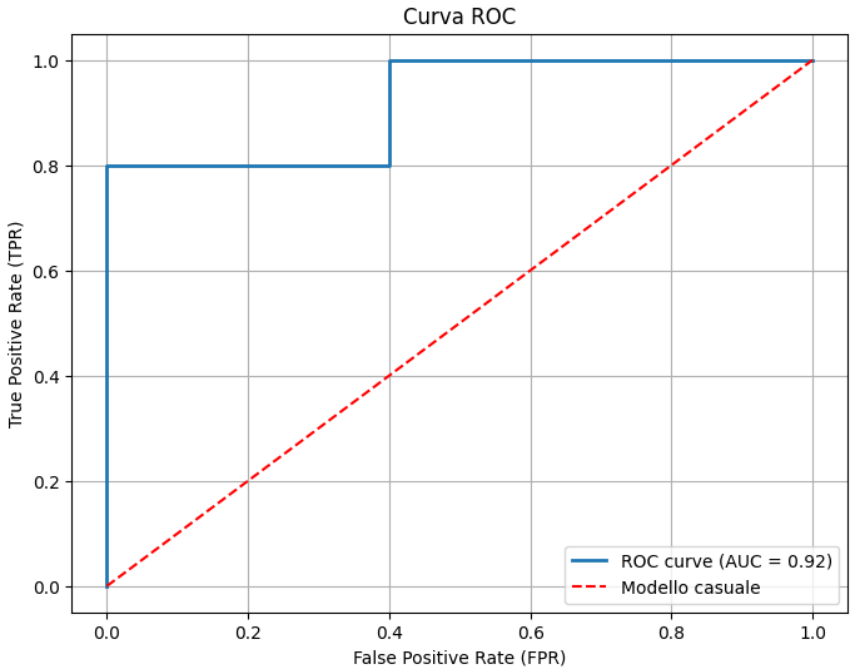


Figura 8.1.: curva ROC.png

8.7.2. Valutazione nei Problemi di Regressione

- **Errore Quadratico Medio (MSE - Mean Square Error)**: L'Errore Quadratico Medio misura la media dei quadrati degli errori tra le previsioni del modello $\hat{y}_i$ e i valori reali y_i. La formula è:

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

dove n è il numero totale di osservazioni.
Il MSE penalizza maggiormente gli errori grandi, rendendolo

particolarmente sensibile ai valori anomali (outlier). È una delle metriche più comuni nei problemi di regressione.

- **Errore Assoluto Medio (MAE - Mean Absolute Error)**:
  L'Errore Assoluto Medio misura la media delle differenze assolute tra le previsioni $\hat{y}_i$\$ e i valori reali y_i. La formula è

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

  dove n è il numero totale di osservazioni. A differenza del MSE, il MAE non eleva al quadrato gli errori, il che lo rende meno sensibile agli outlier. È una scelta utile quando si desidera una valutazione robusta degli errori medi.

- **R² (R-quadrato)**:
  Il coefficiente di determinazione R^2 rappresenta la proporzione della varianza spiegata dal modello rispetto alla varianza totale nei dati. La formula è:

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

  dove $\bar{y}$ è la media dei valori reali.
  Un valore di R^2 vicino a 1 indica che il modello spiega bene la varianza dei dati, mentre un valore vicino a 0 indica il contrario. Tuttavia, questa metrica può risultare fuorviante in alcuni contesti, come in presenza di variabili irrilevanti.

- **Cross-Validation**:
  La **cross-validation** è una tecnica fondamentale per valutare la capacità di generalizzazione di un modello. Tra le varianti più comuni, la **k-fold cross-validation** suddivide il dataset in k sottoinsiemi k-folds. Il modello viene addestrato k volte,

utilizzando ogni volta un fold diverso come set di test e gli altri $k-1$ come set di addestramento. Il punteggio finale è la media dei punteggi calcolati su ciascun fold:

$$\text{Score medio} = \frac{1}{k}\sum_{j=1}^{k}\text{Score}_j$$

Questo metodo riduce il rischio di overfitting e fornisce una stima più affidabile delle prestazioni.

- **Bias e Varianza**:
  Il bilanciamento tra bias e varianza è cruciale nei problemi di regressione. Il **bias** rappresenta l'errore sistematico introdotto da un modello troppo semplice, che non cattura la complessità dei dati (underfitting). La **varianza**, invece, misura quanto il modello è sensibile alle variazioni nei dati di addestramento, portando a overfitting.
  Il tradeoff bias-varianza può essere visualizzato come:

$$\text{Errore totale} = \text{Bias}^2 + \text{Varianza} + \text{Rumore}$$

Dove il **rumore** è l'errore irreducibile presente nei dati. Tecniche come la regolarizzazione (ad esempio, Ridge o Lasso), la scelta di modelli meno complessi o l'ottimizzazione degli iperparametri possono aiutare a trovare il giusto equilibrio.

8.8. Laboratorio Python

8.8.1. Esperimento 1: Predizione della Recidiva su nuovi dati

In questo esperimento Python, procedendo come per l'esperimento in Section 7.12.1 simuleremo un dataset per prevedere la recidiva penale (recidivo o non recidivo) utilizzando un modello di classificazione. Creeremo un trend realistico: ad esempio, chi non ha un lavoro stabile e ha un reato precedente più grave avrà maggiore probabilità di essere recidivo. Infine, esamineremo un nuovo caso e prevederemo la probabilità di recidiva. Per comprendere il codice possiamo pensarlo composto da tre parti:

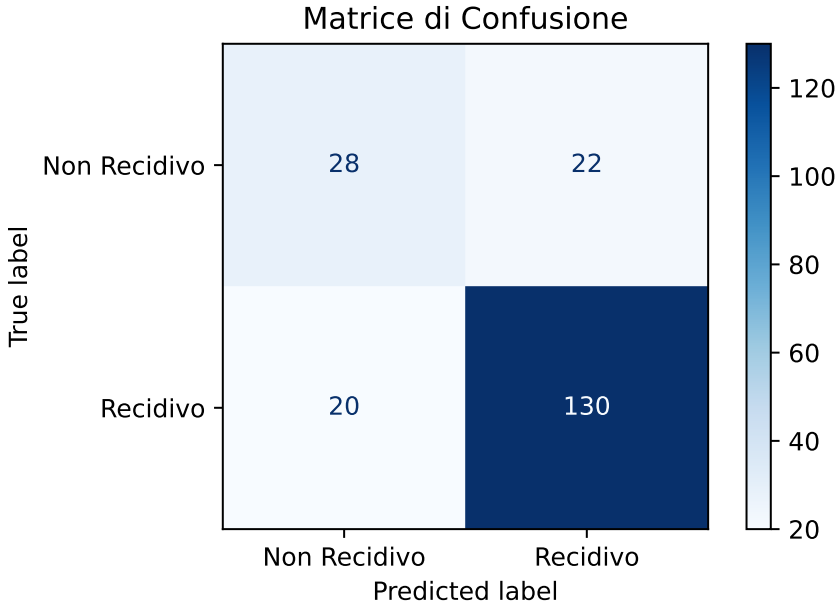
1. **Generazione dei dati**: Simuliamo variabili come età, gravità del reato, lavoro stabile e supporto familiare.
2. **Addestramento del modello**: Usiamo una Random Forest Classifier.
3. **Valutazione del modello**: Misuriamo l'accuratezza e visualizziamo i risultati in una matrice di confusione.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,
 ↪  confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Generazione dei dati simulati
np.random.seed(42)
num_samples = 1000
```

```
eta = np.random.randint(18, 70, num_samples)
gravita_reato = np.random.choice([1, 2, 3],
 ↪  size=num_samples, p=[0.5, 0.3, 0.2])  # 1=furto,
 ↪  2=violenza, 3=omicidio
lavoro_stabile = np.random.choice([0, 1],
 ↪  size=num_samples, p=[0.6, 0.4])
supporto_familiare = np.random.choice([0, 1],
 ↪  size=num_samples, p=[0.4, 0.6])

# Probabilità di recidiva basata su regole
 ↪  realistiche
recidiva_prob = (
    0.4 * (1 - lavoro_stabile) +
    0.3 * (3 - gravita_reato) +
    0.3 * (1 - supporto_familiare)
)
recidiva = (recidiva_prob >
 ↪  np.random.rand(num_samples)).astype(int)

# Creazione del DataFrame
data = pd.DataFrame({
    'eta': eta,
    'gravita_reato': gravita_reato,
    'lavoro_stabile': lavoro_stabile,
    'supporto_familiare': supporto_familiare,
    'recidiva': recidiva
})

# Divisione del dataset
X = data[['eta', 'gravita_reato', 'lavoro_stabile',
 ↪  'supporto_familiare']]
y = data['recidiva']
X_train, X_test, y_train, y_test =
 ↪  train_test_split(X, y, test_size=0.2,
 ↪  random_state=42)
```

```
# Addestramento del modello
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Predizioni e valutazione
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuratezza del modello: {accuracy:.2f}")

# Matrice di confusione
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
 ↪  display_labels=["Non Recidivo", "Recidivo"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Matrice di Confusione")
plt.show()
```

```
Accuratezza del modello: 0.79
```



Adesso, supponiamo di avere un nuovo soggetto con le seguenti caratteristiche:

- **Età**: 30 anni
- **Gravità del reato**: 2 (violenza)
- **Lavoro stabile**: No
- **Supporto familiare**: Sì

Possiamo chiedere al modello di calcolare la probabilità che questo soggetto sia recidivo.

```
# Nuovo soggetto
nuovo_soggetto = pd.DataFrame({
    'eta': [30],
    'gravita_reato': [2],
    'lavoro_stabile': [0],  # No
    'supporto_familiare': [1]  # Sì
```

```
})

# Predizione del rischio di recidiva
rischio_recidiva =
 ↪  model.predict_proba(nuovo_soggetto)[:, 1][0]  #
 ↪  Probabilità di essere recidivo
classe_predetta = model.predict(nuovo_soggetto)[0]
print("dati del nuovo soggetto:")
print(nuovo_soggetto)
print(f"Probabilità di recidiva:
 ↪  {rischio_recidiva:.2f}")
print(f"Classe Predetta: {'Recidivo' if
 ↪  classe_predetta == 1 else 'Non Recidivo'}")
```

```
dati del nuovo soggetto:
   eta  gravita_reato  lavoro_stabile
   supporto_familiare
0   30              2               0
1
Probabilità di recidiva: 1.00
Classe Predetta: Recidivo
```

Ovviamente, questo è solo un esempio di base. Nella pratica, si dovrebbero considerare anche la normalizzazione dei dati, la gestione delle variabili categoriali e l'ottimizzazione degli iperparametri per migliorare le prestazioni. Ciononostante, si invita il lettore a provare a modificare i dati e i modelli per comprendere meglio il funzionamento dei modelli di apprendimento supervisionato.

8.8.2. Esperimento 2: Regressione del Costo di un Immobile

In questo esempio, simuleremo un dataset per prevedere il costo di un immobile. Usiamo un trend realistico: immobili più grandi, in quartieri più costosi e con più bagni avranno un prezzo più alto. Per comprendere il codice possiamo pensarlo composto da tre parti:

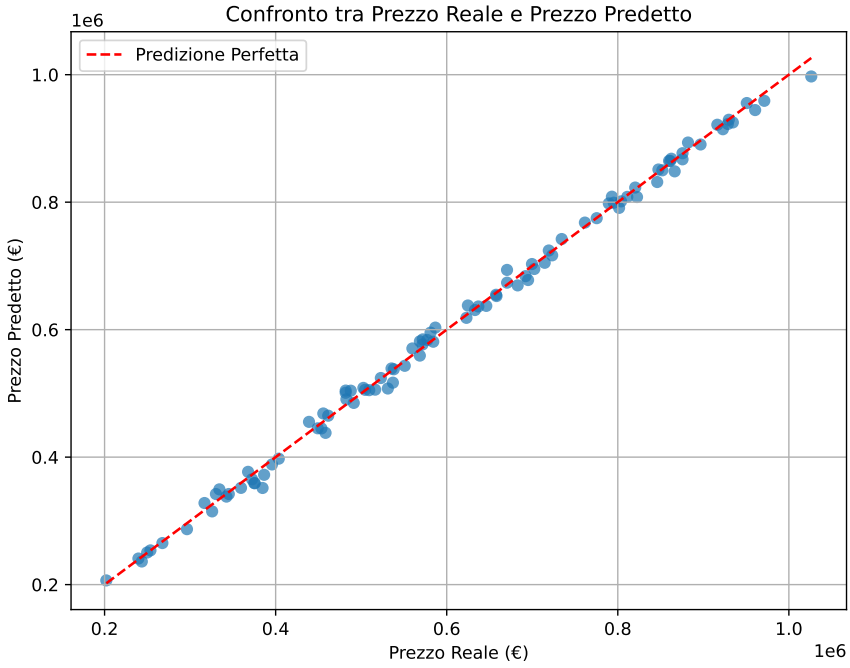
1. **Generazione dei dati**: Simuliamo variabili come superficie, numero di bagni e punteggio del quartiere.
2. **Addestramento del modello**: Usiamo una Random Forest Regressor.
3. **Valutazione del modello**: Visualizziamo i risultati in un grafico di dispersione.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,
 ↪  r2_score

# Generazione dei dati simulati
np.random.seed(42)
num_samples = 500

superficie = np.random.randint(50, 300, num_samples)
 ↪  # Superficie in m²
bagni = np.random.randint(1, 4, num_samples)  #
 ↪  Numero di bagni
quartiere = np.random.randint(1, 6, num_samples)  #
 ↪  Punteggio del quartiere (1-5)

# Prezzo basato su regole realistiche
prezzo = (
    superficie * 3000 +
```

```
    bagni * 10000 +
    quartiere * 20000 +
    np.random.normal(0, 5000, num_samples)  # Rumore
 ↪  casuale
)

# Creazione del DataFrame
data = pd.DataFrame({
    'superficie': superficie,
    'bagni': bagni,
    'quartiere': quartiere,
    'prezzo': prezzo
})

# Divisione del dataset
X = data[['superficie', 'bagni', 'quartiere']]
y = data['prezzo']
X_train, X_test, y_train, y_test =
 ↪  train_test_split(X, y, test_size=0.2,
 ↪  random_state=42)

# Addestramento del modello
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Predizioni e valutazione
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Errore Quadratico Medio (MSE): {mse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
# Grafico di dispersione
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()],
 ↪  [y_test.min(), y_test.max()], '--', color='red',
 ↪  label='Predizione Perfetta')
plt.xlabel("Prezzo Reale (€)")
plt.ylabel("Prezzo Predetto (€)")
plt.title("Confronto tra Prezzo Reale e Prezzo
 ↪  Predetto")
plt.legend()
plt.grid()
plt.show()
```

```
Errore Quadratico Medio (MSE): 121692189.69
R² Score: 1.00
```



Supponiamo di avere un nuovo immobile con le seguenti caratteristiche:

- **Superficie**: 120 m²
- **Numero di bagni**: 2
- **Punteggio del quartiere**: 4

Il modello calcolerà il prezzo stimato per questo immobile.

```
# Nuovo immobile
nuovo_immobile = pd.DataFrame({
    'superficie': [120],
    'bagni': [2],
    'quartiere': [4]
})
```

```
# Predizione del prezzo dell'immobile
prezzo_stimato = model.predict(nuovo_immobile)[0]

# Stampa dei dati del nuovo immobile
print("dati del nuovo immobile:")
print(nuovo_immobile)

# Stampa del prezzo stimato
print(f"Prezzo stimato per il nuovo immobile:
 ↪  €{prezzo_stimato:,.2f}")
```

```
dati del nuovo immobile:
   superficie  bagni  quartiere
0         120      2          4
Prezzo stimato per il nuovo immobile: €438,061.61
```

Occorre ricordare che i dati sono simulati generando dei numeri casuali. Quindi, i legami tra le variabili e le uscite sono altrettanto casuali e potrebbero non essere realistici. Ciononostante, si invita il lettore a provare a modificare i dati e i modelli per comprendere meglio il funzionamento dei modelli di apprendimento supervisionato.

8.9. Esercizi

8.9.1. Esercizio 1: Calcolo delle Metriche di Classificazione

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)

Il lettore calcoli i seguenti valori commentando i risultati ottenuti:

- **Accuratezza (Accuracy)**
- **Precisione (Precision)**
- **Tasso di Recall (Recall)**
- **Tasso di Falso Positivo (False Positive Rate - FPR)**
- **Tasso di Falso Negativo (False Negative Rate - FNR)**

8.9.2. Esercizio 2: Calcolo delle Metriche di regressione

Consideriamo 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

Immobile	Valore Reale (€)	Valore Predetto (€)
4	400,000	420,000
5	350,000	345,000

Il lettore calcoli i seguenti valori commentando i risultati ottenuti:

- **Errore Assoluto Medio (MAE)**
- **Errore Quadratico Medio (MSE)**
- **Errore Quadratico Medio Radice (RMSE)**
- **R^2 (R-quadrato)**

Esercizio 3: Analisi del Rischio di Credito

Una banca vuole sviluppare un modello di apprendimento supervisionato per valutare il rischio di credito dei richiedenti prestiti. L'obiettivo è prevedere se un richiedente sarà in grado di rimborsare il prestito o se c'è un'alta probabilità di inadempienza (default). Per fare ciò la banca ha a disposizione un dataset storico con le informazioni sui clienti e sui prestiti già erogati. **Dataset (Simulato):**

- **Età** : Età del richiedente (valore numerico, in anni).
- **Reddito Annuale** : Reddito annuo del richiedente (valore numerico, in euro).
- **Anni di Impiego** : Anni di impiego del richiedente (valore numerico).
- **Importo del Prestito** : Importo del prestito richiesto (valore numerico, in euro).
- **Punteggio di Credito** : Punteggio di credito del richiedente (valore numerico, tra 300 e 850).
- **Presenza di Garante** : Se il richiedente ha presentato un garante (valore binario: 0 = No, 1 = Sì).
- **Stato di Inadempienza** : Se il richiedente è andato in default sul prestito (valore binario: 0 = No, 1 = Sì).

Il lettore è invitato a:

- spiegare se si tratta di un problema di apprendimento supervisionato o non supervisionato.
- indichi il tipo di problema di apprendimento supervisionato è (classificazione o regressione).
- indichi qual è la variabile target (o etichetta) in questo problema?
- indichi quali sono le feature?

9. Apprendimento Non Supervisionato

9.1. Introduzione

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.

9.2. 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 (MacQueen 1967).
- **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 (Rokach and Maimon 2005).
- **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 (Ester et al. 1996).

9.2.1. Confronto tra Metodi

Metodo	Vantaggi	Svantaggi
K-means	Semplice da implementare; efficiente su grandi dataset	Sensibile alla scelta di k e ai valori iniziali dei centroidi
Hierarchical Clustering	Rappresentazione visiva con dendrogrammi; non richiede k iniziale	Poco scalabile su grandi dataset
DBSCAN	Rileva cluster di forma arbitraria; gestisce outlier	Richiede tuning accurato dei parametri eps e minPts

9.3. 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 (Hotelling 1933).

- **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 (Van der Maaten and Hinton 2008).

- **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 (Hinton and Salakhutdinov 2006).

9.4. 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 (Newman 2004).

- **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**, identificando temi ricorrenti o gruppi di casi simili.
- La **scoperta di pattern** nei dati dei casi giudiziari, come tipologie di reati o fattori comuni nei procedimenti legali.
- La **segmentazione dei casi giudiziari**, utile per organizzare i casi in categorie omogenee.
- Il **rilevamento delle anomalie** nei contratti legali o nelle transazioni finanziarie.

Queste tecniche forniscono strumenti potenti per esplorare e comprendere i dati in modo più profondo, senza la necessità di etichette predefinite.

9.5. Laboratorio Python

9.5.1. Esperimento 1: Segmentazione di documenti legali con K-means

In questo esperimento Python, utilizziamo l'algoritmo K-means per raggruppare documenti legali basandoci sulla frequenza delle parole chiave presenti nei documenti. Si ipotizza che i documenti legali possano appartenere a categorie specifiche, come Contratti, Reati, Proprietà Privata, Assicurazioni, e Contratti di Lavoro. In particolare, si ipotizza che siano presenti 3 categorie principali. Il numero di cluster (K) è impostato a 3.

Il funzionamento del semplice codice che segue può essere descritto come segue:

1. Si caricano le stop word della lingua italiana in modo che il codice possa rimuovere correttamente le parole comuni in italiano come "il", "di", "la", migliorando la qualità del clustering K Stop Words Italiane;
2. I documenti legali sono trasformati in vettori numerici usando la tecnica **TF-IDF**, che calcola l'importanza delle parole in ciascun documento.
3. L'algoritmo K-means raggruppa i documenti in 3 cluster (ad esempio, Contratti, Reati, Proprietà Privata).
4. Ogni documento viene assegnato a un cluster in base alla somiglianza del suo contenuto con altri documenti.

```
from sklearn.feature_extraction.text import
 ↪  TfidfVectorizer
from sklearn.cluster import KMeans
from nltk.corpus import stopwords
import nltk

# Scarica le stop words italiane
nltk.download('stopwords')
stop_words_italiane = stopwords.words('italian')

# Dati simulati: estratti di documenti legali
documenti_legali = [
    "Il contratto di locazione è regolato dal Codice
     ↪  Civile.",
    "Il reato di furto è punito secondo l'articolo
     ↪  624.",
    "La proprietà privata è garantita dalla
     ↪  Costituzione.",
    "L'assicurazione copre i danni derivanti da
     ↪  incidenti stradali.",
    "Il contratto di lavoro subordinato deve
     ↪  rispettare le norme vigenti.",
    "Il Codice Penale stabilisce le pene per i reati
     ↪  contro la persona. Le pene sono determinate
     ↪  dalla gravità del reato.",
    "Le controversie contrattuali sono risolte
     ↪  tramite arbitrato.",
]

# Trasformazione dei documenti in rappresentazioni
 ↪  numeriche (TF-IDF)
vectorizer =
 ↪  TfidfVectorizer(stop_words=stop_words_italiane)
```

```
X = vectorizer.fit_transform(documenti_legali)

# Applicazione di K-means
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Etichette dei cluster
labels = kmeans.labels_

# Visualizzazione dei risultati
for i, documento in enumerate(documenti_legali):
    print(f"Documento: {documento}")
    print(f"Appartiene al cluster: {labels[i]}\n")
```

```
Documento: Il contratto di locazione è regolato dal
Codice Civile.
Appartiene al cluster: 2

Documento: Il reato di furto è punito secondo
l'articolo 624.
Appartiene al cluster: 1

Documento: La proprietà privata è garantita dalla
Costituzione.
Appartiene al cluster: 0

Documento: L'assicurazione copre i danni derivanti
da incidenti stradali.
Appartiene al cluster: 0

Documento: Il contratto di lavoro subordinato deve
rispettare le norme vigenti.
Appartiene al cluster: 2
```

```
Documento: Il Codice Penale stabilisce le pene per i
reati contro la persona. Le pene sono determinate
dalla gravità del reato.
Appartiene al cluster: 1

Documento: Le controversie contrattuali sono risolte
tramite arbitrato.
Appartiene al cluster: 0

[nltk_data] Downloading package stopwords to
[nltk_data]
C:\Users\lcapitanio\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already
up-to-date!
C:\Users\lcapitanio\AppData\Local\Packages\PythonSoftwareFoun
UserWarning:

Could not find the number of physical cores for the
following reason:
[WinError 2] Impossibile trovare il file specificato
Returning the number of logical cores instead. You
can silence this warning by setting
LOKY_MAX_CPU_COUNT to the number of cores you want
to use.

  File
  "C:\Users\lcapitanio\AppData\Local\Packages\PythonSoftwareF
  line 257, in _count_physical_cores
    cpu_info = subprocess.run(
               ^^^^^^^^^^^^^^^

  File "C:\Program
  Files\WindowsApps\PythonSoftwareFoundation.Python.3.12_3.12
  line 548, in run
```

```
  with Popen(*popenargs, **kwargs) as process:
       ^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.12_3.12.2
line 1026, in __init__
  self._execute_child(args, executable,
  preexec_fn, close_fds,
File "C:\Program
Files\WindowsApps\PythonSoftwareFoundation.Python.3.12_3.12.2
line 1538, in _execute_child
  hp, ht, pid, tid =
  _winapi.CreateProcess(executable, args,

                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

9.5.2. Esperimento 2: Identificazione di anomalie nei dati giudiziari con DBSCAN

In questo epserimento useremo l'algoritmo DBSCAN per analizzare un dataset multivariato simulato in ambito giuridico. L'obiettivo è individuare pattern o anomalie nei dati relativi a sentenze legali. Il funzionamento del codice che segue può essere descritto come segue:

1. Viene generato un dataset simulato contenente variabili come numero di testimoni, durata dei processi, gravità del reato, ecc.
2. L'algoritmo DBSCAN viene applicato per identificare cluster di casi simili e individuare eventuali anomalie o outlier. DBSCAN assegna il valore `-1` ai punti considerati outlier.
3. I risultati vengono visualizzati graficamente in 2D (numero di testimoni contro durata dei processi), con cluster distinti identificati da colori diversi, per una comprensione visiva dei

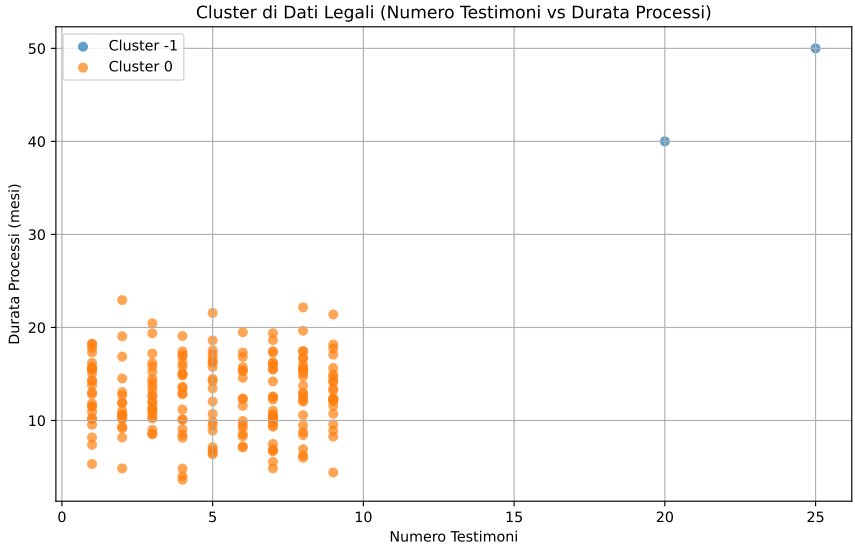
cluster e delle anomalie. In particolare, il cluster -1 è quello composto dalle anomalie.

```
import numpy as np
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Generazione di dati simulati: caratteristiche di
 ↪  sentenze legali
np.random.seed(42)

# Creazione di variabili
numero_testimoni = np.random.randint(1, 10,
 ↪  size=200)  # Numero di testimoni per caso
durata_processi = np.random.normal(loc=12, scale=4,
 ↪  size=200)  # Durata dei processi in mesi
gravita_reato = np.random.randint(1, 5, size=200)  #
 ↪  Gravità del reato (1=leggera, 4=grave)

# Creazione di outlier
outlier_testimoni = [20, 25]
outlier_durata = [40, 50]
outlier_gravita = [5, 5]

# Inserimento degli outlier nei dati
numero_testimoni = np.append(numero_testimoni,
 ↪  outlier_testimoni)
durata_processi = np.append(durata_processi,
 ↪  outlier_durata)
gravita_reato = np.append(gravita_reato,
 ↪  outlier_gravita)
```

```
# Combinazione delle variabili in un DataFrame
data = pd.DataFrame({
    'Numero_Testimoni': numero_testimoni,
    'Durata_Processi': durata_processi,
    'Gravita_Reato': gravita_reato
})

# Standardizzazione dei dati
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Applicazione dell'algoritmo DBSCAN
dbscan = DBSCAN(eps=1.5, min_samples=5)  # Parametri
 ↪  regolabili
labels = dbscan.fit_predict(data_scaled)

# Aggiunta delle etichette al DataFrame
data['Cluster'] = labels

# Visualizzazione dei risultati
print("Esempio di cluster individuati:")
print(data.head())

# Visualizzazione in 2D (numero testimoni vs durata
 ↪  processi)
plt.figure(figsize=(10, 6))
for cluster in np.unique(labels):
    cluster_data = data[data['Cluster'] == cluster]
    plt.scatter(cluster_data['Numero_Testimoni'],
 ↪  cluster_data['Durata_Processi'], label=f"Cluster
 ↪  {cluster}", alpha=0.7)

plt.title("Cluster di Dati Legali (Numero Testimoni
 ↪  vs Durata Processi)")
```

```
plt.xlabel("Numero Testimoni")
plt.ylabel("Durata Processi (mesi)")
plt.legend()
plt.grid()
plt.show()
```

```
Esempio di cluster individuati:
   Numero_Testimoni  Durata_Processi  Gravita_Reato
   Cluster
0                 7        10.446942              3
0
1                 4         8.457951              3
0
2                 8        10.573020              1
0
3                 5        14.224487              4
0
4                 7        16.175442              2
0
```



Con questo approccio, è possibile identificare sia i cluster naturali nei dati sia eventuali anomalie (outlier), che potrebbero rappresentare sentenze particolarmente atipiche o significative.

9.6. Esercizi

9.6.1. Esercizio 1: Analisi delle variabili rilevanti per il clustering dei casi giudiziari

Un'agenzia governativa sta analizzando un dataset relativo ai casi giudiziari per identificare gruppi omogenei di processi. Le variabili disponibili includono:

- Durata del processo (in mesi).
- Tipologia di reato (furto, frode, omicidio, ecc.).
- Numero di testimoni ascoltati.

- Gravità della pena prevista (da 1 a 5).

Domande:

1. Quali variabili pensi siano più rilevanti per identificare cluster di processi simili? Motiva la tua scelta.
2. Spiega come la scelta delle variabili potrebbe influenzare il risultato del clustering.
3. Quali misure adotteresti per assicurarti che i dati siano ben preparati prima di applicare un algoritmo di clustering?

9.6.2. Esercizio 2: Interpretazione dei risultati del clustering e loro applicazione nel sistema giudiziario

Immagina di aver applicato un algoritmo di clustering a un dataset giuridico e di aver ottenuto tre cluster con le seguenti caratteristiche medie:

- **Cluster 1**: Durata del processo breve (6 mesi), reati di bassa gravità (furto), pochi testimoni (1-2).
- **Cluster 2**: Durata media (18 mesi), reati di gravità moderata (frode), testimoni multipli (5-10).
- **Cluster 3**: Durata lunga (36 mesi), reati gravi (omicidio), molti testimoni (15-20).

Domande:

1. Quali interpretazioni puoi dare ai tre cluster? Ad esempio, rappresentano categorie specifiche di processi?
2. Come potrebbero questi risultati essere utilizzati per migliorare l'efficienza del sistema giudiziario?
3. Se ci fossero casi che non si adattano a nessun cluster, come li gestiresti?

9.6.3. Esercizio 3: Riduzione della dimensionalità per semplificare l'analisi dei dati legali

Un ricercatore sta analizzando un dataset di documenti legali che contiene centinaia di variabili, come parole chiave, lunghezza del documento e numero di citazioni. Il ricercatore vuole semplificare il dataset utilizzando una tecnica di riduzione della dimensionalità come **PCA** (Principal Component Analysis).

Domande:

1. Quali vantaggi offre l'uso di una tecnica come PCA in questo caso?
2. Dopo aver ridotto la dimensionalità dei dati, come valuteresti se l'informazione essenziale è stata preservata?
3. In che modo l'analisi dei dati ridotti potrebbe facilitare l'identificazione di pattern nei documenti legali?

10. Bias

10.1. Introduzione

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 fisiche, sulle persone giuridiche e sulla società in generale. Il bias può portare a previsioni errate e ingiuste, perpetuando disuguaglianze sociali e legali. Quindi, la comprensione e la gestione del bias sono essenziali per garantire la trasparenza e la giustizia nei sistemi legali e sociali.

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.

10.2. 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.

- **Bias di Conferma**: Questo tipo di bias emerge quando i dati o le caratteristiche selezionate per il modello confermano preconcetti o ipotesi preesistenti. Ad esempio, un modello per la concessione del credito che utilizza dati storici potrebbe favorire inconsciamente gli individui di una determinata etnia o genere.
- **Bias di Sopravvivenza**: 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.
- **Bias Sistemico**: Riflette le disuguaglianze o le discriminazioni già presenti nei dati storici e nei sistemi sociali. Ad esempio, un modello che predice la recidiva basato su dati storici che riflettono pratiche discriminatorie della polizia potrebbe perpetuare tali discriminazioni.
- **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, un algoritmo di scoring creditizio ottimizzato esclusivamente per ridurre i tassi di insolvenza potrebbe penalizzare ingiustamente gruppi demografici con meno accesso al credito.

10.3. Cause e Impatti del Bias

10.3.1. Cause del Bias

1. **Dataset Non Rappresentativi**: Uno dei motivi principali del bias è l'uso di dataset non rappresentativi della popolazione target. 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.
2. **Scelte di Modellazione**: Le decisioni prese durante la fase di modellazione, come la selezione delle caratteristiche o la scelta dell'algoritmo, possono introdurre bias. Per esempio, includere variabili come il "quartiere di residenza" in un modello potrebbe introdurre bias socio-economici.
3. **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 potrebbe generare più arresti in quelle aree, creando un ciclo che perpetua il bias.

10.3.2. Impatti del Bias

1. **Discriminazione**: Un modello biased può perpetuare o amplificare disuguaglianze esistenti. Ad esempio, se un modello di scoring creditizio discrimina contro minoranze etniche, potrebbe negare l'accesso al credito a persone meritevoli.
2. **Perdita di Fiducia**: Sistemi percepiti come ingiusti o discriminatori possono compromettere la fiducia del pubblico.

3. **Conseguenze Legali**: L'uso di modelli biased in decisioni legali può portare a contenziosi e danni reputazionali per le istituzioni, oltre a violare normative sulla non discriminazione.

10.4. Tecniche di Mitigazione del Bias

1. **Raccolta e Preparazione dei Dati**: Assicurarsi che il dataset sia rappresentativo della popolazione target.
2. **Pre-processing dei Dati**: Applicare tecniche per ridurre il bias prima dell'addestramento, come la rimozione di caratteristiche correlate al bias (ad esempio, genere o etnia).
3. **Modellazione In-process**: Utilizzare regolarizzazioni che penalizzano il modello se sfrutta caratteristiche correlate al bias.
4. **Post-processing**: Correggere il bias nelle previsioni del modello, ad esempio attraverso la calibrazione delle probabilità predette.
5. **Monitoraggio e Aggiornamento Continuo**: Valutare e aggiornare periodicamente i modelli per garantire che il bias non peggiori.
6. **Analisi di Impatto e Auditing**: Condurre audit regolari per identificare e correggere bias, coinvolgendo esperti tecnici ed etici.

10.5. 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 rispettino i principi di equità e non discriminazione, minimizzando il rischio di perpetuare disuguaglianze e promuovendo decisioni più giuste e trasparenti.

10.6. Laboratorio in Python

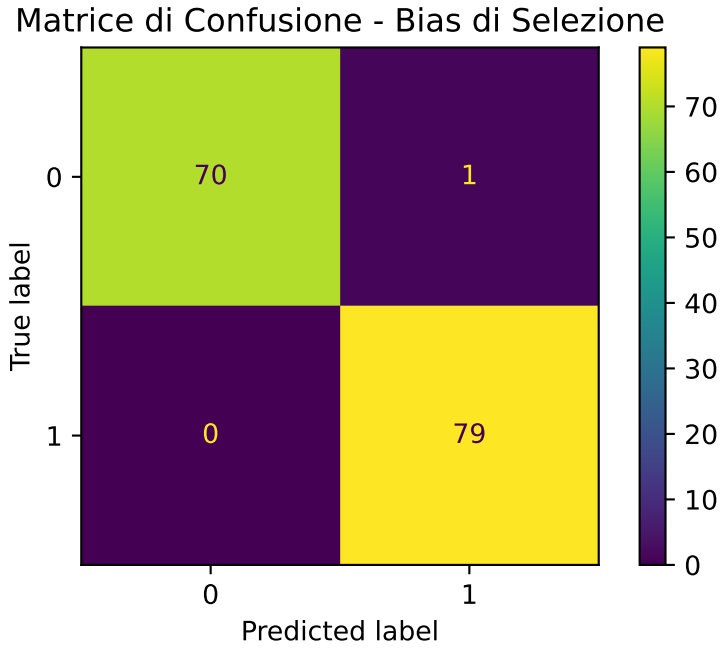
10.6.1. Esperimento 1: Bias di Selezione

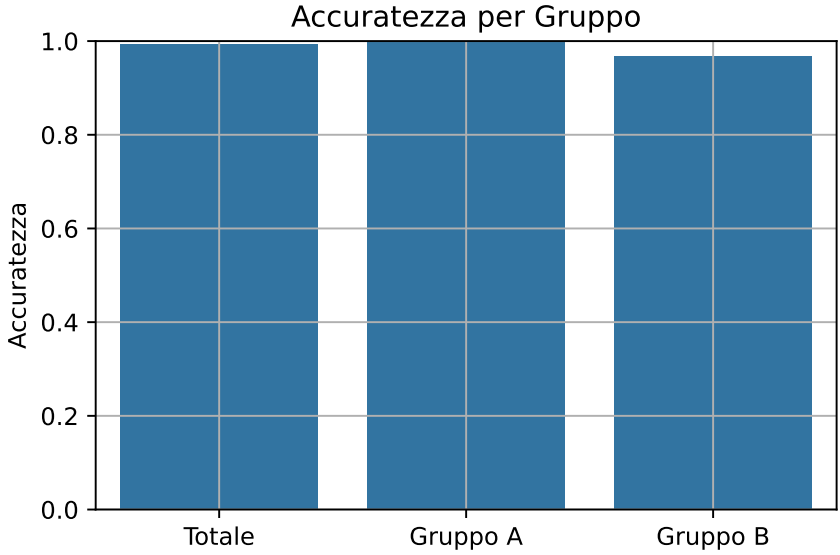
Simuliamo un dataset dove la popolazione rappresentata è squilibrata tra due gruppi. Addestriamo un modello e analizziamo l'accuratezza per ciascun gruppo.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
 ↪  confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
import seaborn as sns

# Simulazione di un dataset con bias di selezione
np.random.seed(42)
n_samples = 500

# Gruppo A (es. uomini)
X_A = np.random.normal(loc=50, scale=10,
 ↪  size=(int(n_samples * 0.8), 1))  # più
 ↪  rappresentato
y_A = (X_A.flatten() > 50).astype(int)  # outcome
 ↪  dipende dal valore
```

```
# Gruppo B (es. donne), sottorappresentato
X_B = np.random.normal(loc=50, scale=10,
 ↪  size=(int(n_samples * 0.2), 1))
y_B = (X_B.flatten() > 50).astype(int)

# Combinazione dei dati
X = np.vstack((X_A, X_B))
y = np.concatenate((y_A, y_B))
groups = np.array(['A'] * len(X_A) + ['B'] *
 ↪  len(X_B))

# Divisione del dataset
X_train, X_test, y_train, y_test, groups_train,
 ↪  groups_test = train_test_split(X, y, groups,
 ↪  test_size=0.3, random_state=42)

# Addestramento modello
model = LogisticRegression()
model.fit(X_train, y_train)

# Predizioni
y_pred = model.predict(X_test)

# Accuratezza complessiva
acc = accuracy_score(y_test, y_pred)

# Accuratezza per gruppo
acc_A = accuracy_score(y_test[groups_test == 'A'],
 ↪  y_pred[groups_test == 'A'])
acc_B = accuracy_score(y_test[groups_test == 'B'],
 ↪  y_pred[groups_test == 'B'])
```

```
# Matrice di confusione
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Matrice di Confusione - Bias di
 ↪  Selezione")
plt.show()

# Visualizzazione delle accuratezze
sns.barplot(x=['Totale', 'Gruppo A', 'Gruppo B'],
 ↪  y=[acc, acc_A, acc_B])
plt.title('Accuratezza per Gruppo')
plt.ylabel('Accuratezza')
plt.ylim(0, 1)
plt.grid(True)
plt.show()

acc, acc_A, acc_B
```


Matrice di Confusione - Bias di Selezione
70
1
0
79
True label
Predicted label
0
1
70
60
50
40
30
20
10
0


Accuratezza per Gruppo
Accuratezza
1.0
0.8
0.6
0.4
0.2
0.0
Totale
Gruppo A
Gruppo B

I risultati dell'esperimento sul **bias di selezione** sono i seguenti:

- **Accuratezza complessiva**: 99.3%
- **Accuratezza sul Gruppo A** (maggiormente rappresentato): 100%
- **Accuratezza sul Gruppo B** (sottorappresentato): 96.8%

Questo esperimento evidenzia il pericolo del **bias di selezione**, un problema comune nei sistemi di classificazione. Anche se il modello ha un'elevata accuratezza complessiva, la sua performance sul gruppo sottorappresentato è significativamente inferiore. Questo dimostra come un modello possa essere ingiusto verso alcune categorie, anche se sembra essere accurato in generale.

10.6.2. Esperimento 2: Bias Sistemico

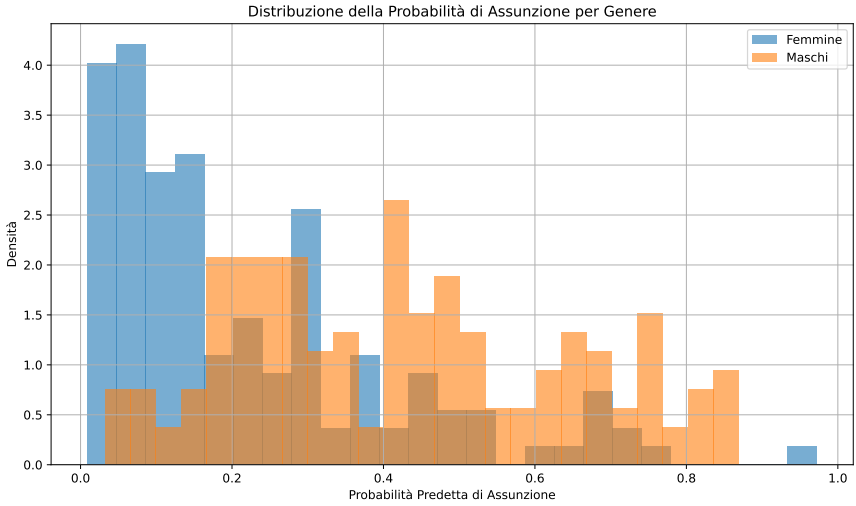
L'obiettivo di questo esperimento è dimostrare l'effetto di un bias sistemico di genere in un modello di selezione del personale. A questo scopo simuliamo un processo di selezione del personale in cui, a parità di esperienza e età, **le donne vengono penalizzate** sistematicamente rispetto agli uomini. Usiamo un dataset artificiale in cui il genere influisce negativamente sulla probabilità di assunzione, creando così un **bias sistemico**.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Generazione dati simulati
```

```
np.random.seed(42)
n = 1000
esperienza = np.random.normal(5, 2, n)
eta = np.random.normal(35, 5, n)
genere = np.random.choice([0, 1], size=n)  # 0 =
 ↪  femmina, 1 = maschio

# Bias sistemico: penalizzazione delle donne
 ↪  (genere=0)
logits = 0.5 * esperienza + 0.1 * eta + 1.0 * genere
 ↪  - 7.5
prob_assunzione = 1 / (1 + np.exp(-logits))
assunto = np.random.binomial(1, prob_assunzione)

# Dataset
df = pd.DataFrame({
    'esperienza': esperienza,
    'eta': eta,
    'genere': genere,
    'assunto': assunto
})

# Split e normalizzazione
X = df[['esperienza', 'eta', 'genere']]
y = df['assunto']
X_train, X_test, y_train, y_test =
 ↪  train_test_split(X, y, test_size=0.3,
 ↪  random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Modello
model = LogisticRegression()
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

# Classificazione report
report = classification_report(y_test, y_pred,
 ↪  output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Grafico: distribuzione delle probabilità predette
 ↪  per maschi e femmine
prob_pred = model.predict_proba(X_test_scaled)[:, 1]
X_test_with_probs = pd.DataFrame(X_test_scaled,
 ↪  columns=['esperienza', 'eta', 'genere'])
X_test_with_probs['prob_assunto'] = prob_pred
X_test_with_probs['genere'] =
 ↪  X_test['genere'].values

# Plot
plt.figure(figsize=(10, 6))
for g, label in zip([0, 1], ['Femmine', 'Maschi']):
    subset =
 ↪  X_test_with_probs[X_test_with_probs['genere'] ==
 ↪  g]
    plt.hist(subset['prob_assunto'], bins=25,
 ↪  alpha=0.6, label=label, density=True)
plt.xlabel('Probabilità Predetta di Assunzione')
plt.ylabel('Densità')
plt.title('Distribuzione della Probabilità di
 ↪  Assunzione per Genere')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```



1. **Importazione delle librerie**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import ConfusionMatrixDisplay,
 ↪  classification_report
from sklearn.model_selection import train_test_split
```

Queste librerie permettono di:

- creare e gestire dati (`numpy`, `pandas`)
- visualizzare grafici (`matplotlib`)

- creare e valutare un modello di regressione logistica (`sklearn`)

---

2. **Generazione del dataset simulato**

```
np.random.seed(42)
n = 1000
genere = np.random.binomial(1, 0.5, n)  # 0=f, 1=m
esperienza = np.random.normal(5 + genere*1.5, 1, n)
 ↪  # maschi con +1.5 anni
eta = np.random.normal(35, 5, n)
```

Qui generiamo 1000 candidati:

- `genere`: 0 = femmina, 1 = maschio, distribuiti equamente.
- `esperienza`: in media, i maschi hanno più esperienza (bias indotto).
- `età`: distribuita normalmente.

```
bias = -0.8 * (1 - genere)  # penalizzazione per le
 ↪  donne
logits = 0.3*esperienza + 0.05*eta + bias
prob = 1 / (1 + np.exp(-logits))
assunto = np.random.binomial(1, prob)
```

- Calcoliamo la **probabilità di assunzione** come combinazione lineare delle variabili, con un **bias penalizzante per le donne**.
- La variabile `assunto` simula se il candidato viene assunto.

3. **Addestramento del modello**

```
X = np.column_stack((esperienza, eta, genere))
y = assunto
X_train, X_test, y_train, y_test =
 ↪  train_test_split(X, y, test_size=0.3,
 ↪  random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

- Addestriamo un modello di **regressione logistica** per predire l'assunzione.
- Usiamo il **genere** come input, quindi il modello può apprendere e replicare il bias.

4. **Visualizzazione del bias**

```
X_df = pd.DataFrame(X_test, columns=["esperienza",
 ↪  "eta", "genere"])
X_df["prob_pred"] = model.predict_proba(X_test)[:,1]
X_df["genere"] = X_df["genere"].astype(int)

plt.figure(figsize=(8, 6))
for g in [0, 1]:
    subset = X_df[X_df["genere"] == g]
    label = "Femmine" if g == 0 else "Maschi"
    plt.hist(subset["prob_pred"], bins=20,
 ↪  alpha=0.5, label=label, density=True)

plt.title("Distribuzione delle probabilità predette
 ↪  di assunzione per genere")
plt.xlabel("Probabilità predetta di assunzione")
plt.ylabel("Densità")
```

```
plt.legend()
plt.grid(True)
plt.show()
```

- **Istogramma** delle probabilità predette per ciascun genere.
- Mostra chiaramente che i **maschi** hanno **più probabilità di essere assunti**, anche a parità di altre condizioni.

Questo esperimento dimostra visivamente e quantitativamente:

- l'effetto del **bias sistemico**,
- come esso possa essere **appreso e amplificato** da un modello,
- perché sia cruciale **rimuovere variabili sensibili** o applicare **strategie di mitigazione**.

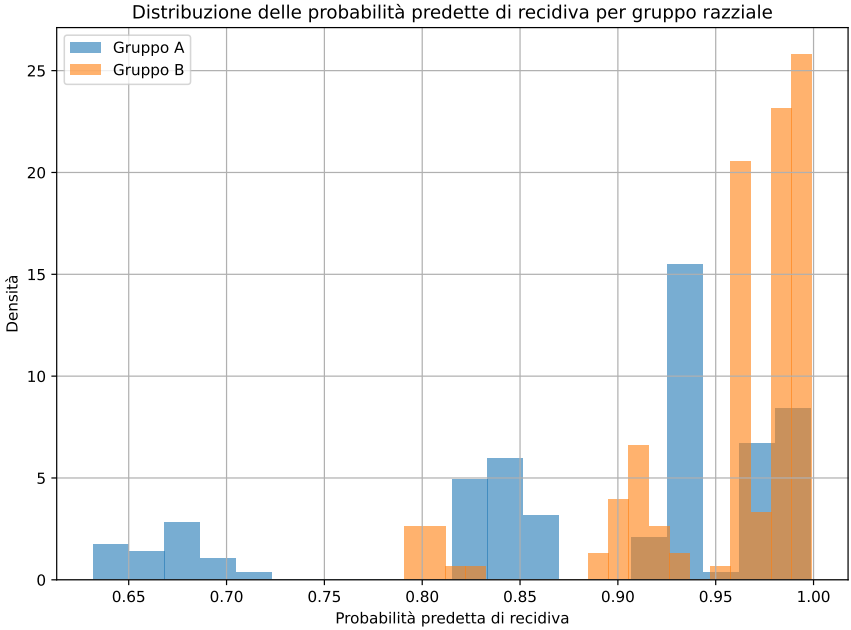
10.6.3. Esperimento 2: Bias di razza nella valutazione del rischio di recidiva

Obiettivo di questo esperimento è dimostrare come un modello predittivo possa riflettere o amplificare bias razziali analizzando la distribuzione delle probabilità predette per la recidiva in due gruppi demografici distinti.

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

# 1. Simulazione dataset giustizia penale con
 ↪  possibile bias razziale
```

```
np.random.seed(42)
n = 1000
# Variabile sensibile: razza (0=gruppo A, 1=gruppo
 ↪  B)
razza = np.random.binomial(1, 0.5, n)

# Caratteristiche: numero precedenti penali, età
precedenti = np.random.poisson(2 + razza * 0.5, n)
 ↪  # il gruppo B ha in media più precedenti
eta = np.random.normal(35, 5, n)

# Probabilità "vera" di recidiva senza bias
logits_veri = 0.8 * precedenti + 0.02 * eta
prob_vera = 1 / (1 + np.exp(-logits_veri))
recidiva_reale = np.random.binomial(1, prob_vera)

# Sistema giudiziario introduce un bias implicito
 ↪  (più severo con gruppo B)
bias = 0.7 * razza
logits_predetti = logits_veri + bias
prob_predetta = 1 / (1 + np.exp(-logits_predetti))
recidiva_predetta = np.random.binomial(1,
 ↪  prob_predetta)

# 2. Addestramento modello su dati "biasati"
X = np.column_stack((precedenti, eta, razza))
y = recidiva_predetta
X_train, X_test, y_train, y_test =
 ↪  train_test_split(X, y, test_size=0.3,
 ↪  random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
probs = model.predict_proba(X_test)[:, 1]

# 3. Analisi dell'equità: confronto distribuzioni
 ↪  delle probabilità predette per gruppo
X_test_df = pd.DataFrame(X_test,
 ↪  columns=["precedenti", "eta", "razza"])
X_test_df["prob_pred"] = probs
X_test_df["razza"] = X_test_df["razza"].astype(int)

# 4. Grafico della distribuzione predetta per razza
plt.figure(figsize=(8, 6))
for r in [0, 1]:
    subset = X_test_df[X_test_df["razza"] == r]
    label = "Gruppo A" if r == 0 else "Gruppo B"
    plt.hist(subset["prob_pred"], bins=20,
 ↪  alpha=0.6, label=label, density=True)

plt.title("Distribuzione delle probabilità predette
 ↪  di recidiva per gruppo razziale")
plt.xlabel("Probabilità predetta di recidiva")
plt.ylabel("Densità")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# 5. Rapporto di classificazione
report = classification_report(y_test, y_pred,
 ↪  target_names=["No recidiva", "Recidiva"])
report
```




```
C:\Users\lcapitanio\AppData\Local\Packages\PythonSoftwareFounda
UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in
labels with no predicted samples. Use
`zero_division` parameter to control this behavior.

C:\Users\lcapitanio\AppData\Local\Packages\PythonSoftwareFounda
UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in
labels with no predicted samples. Use
`zero_division` parameter to control this behavior.

C:\Users\lcapitanio\AppData\Local\Packages\PythonSoftwareFounda
UndefinedMetricWarning:
```

```
Precision is ill-defined and being set to 0.0 in
labels with no predicted samples. Use
`zero_division` parameter to control this behavior.

'              precision    recall  f1-score
support\n\n No recidiva       0.00      0.00
0.00        26\n    Recidiva       0.91      1.00
0.95       274\n\n    accuracy
0.91       300\n   macro avg       0.46      0.50
0.48       300\nweighted avg       0.83      0.91
0.87       300\n'
```

Il codice Python simula un dataset di giustizia penale con possibile bias razziale. Il modello predittivo apprende e amplifica questo bias, mostrando come la distribuzione delle probabilità predette sia diversa per i due gruppi.

1. **Simulazione dei dati**

Abbiamo simulato un dataset di **1.000 soggetti** nel contesto della giustizia penale, con le seguenti caratteristiche:

- **Razza** (`0=Gruppo A`, `1=Gruppo B`)
- **Numero di precedenti penali**
- **Età**

La **variabile sensibile** è la razza. Abbiamo introdotto un *bias implicito* che aumenta il rischio predetto di recidiva per il gruppo B, anche a parità di condizioni con il gruppo A.

2. **Addestramento del modello**

È stato addestrato un modello di **regressione logistica** sui dati biasati per simulare l'apprendimento in un ambiente non equo.

3. **Analisi dell'equità**

Abbiamo calcolato le **probabilità predette** di recidiva per ogni soggetto nel test set e confrontato graficamente le distribuzioni nei due gruppi razziali.

4. **Risultati**

Il grafico mostra chiaramente che il **gruppo B riceve sistematicamente predizioni di rischio più alte**, anche a parità di precedenti e di età. Questo è un segnale diretto di **bias sistemico** nel processo predittivo.

5. **Valutazione delle prestazioni**

Dal report di classificazione:

- Il modello ha una **accuratezza del 91%**, ma è totalmente sbilanciato: **non predice mai la classe "No recidiva"** (precision = 0.00, recall = 0.00).
- Questo è un esempio perfetto di come un modello possa sembrare "accurato" ma essere **profondamente ingiusto**.

L'esperimento dimostra che:

- Il bias può essere introdotto **nei dati**, non necessariamente nel modello.
- È fondamentale **analizzare le predizioni per sottogruppi** (e.g., gruppi razziali, genere, età).
- Anche modelli con alta accuratezza possono **comportarsi in modo discriminatorio**, e le metriche aggregate possono mascherare queste disuguaglianze.

10.7. Esercizi

10.7.1. Esercizio 1: Identificazione dei Bias nei Dati

Obiettivo: Riconoscere le diverse tipologie di bias in un contesto giuridico.

Scenario: Un tribunale sta implementando un sistema di **intelligenza artificiale** per prevedere la probabilità di recidiva degli imputati. Il modello è stato addestrato su dati storici, che includono informazioni su età, genere, reati precedenti e quartiere di residenza.

Dopo alcuni mesi di utilizzo, emergono alcune anomalie:

- Il modello assegna un rischio di recidiva più alto agli imputati provenienti da determinati quartieri.
- Le donne ricevono punteggi di recidiva inferiori rispetto agli uomini, anche a parità di altri fattori.
- Gli imputati di età inferiore ai 30 anni vengono spesso classificati come ad alto rischio di recidiva, anche se alcuni non hanno precedenti gravi.

Domande:

1. Quali tipi di bias sono presenti in questo modello? (Selezione, conferma, sopravvivenza, sistemico, algoritmico)
2. Quali potrebbero essere le cause di questi bias nei dati storici utilizzati per l'addestramento?
3. Quali soluzioni potrebbero essere adottate per mitigare questi bias?

10.7.2. Esercizio 2: Bias e Implicazioni Giuridiche

Obiettivo: Analizzare le conseguenze legali ed etiche dei bias nei sistemi di intelligenza artificiale.

Scenario: Un'azienda sviluppa un software di **selezione del personale** basato su machine learning, addestrato con dati storici dell'azienda stessa. Dopo l'implementazione, emerge che il software tende a selezionare prevalentemente uomini per le posizioni manageriali.

Alcuni dipendenti sollevano preoccupazioni e accusano l'azienda di **discriminazione indiretta**. Un comitato interno analizza i dati e scopre che, negli ultimi 10 anni, il 75% dei dirigenti assunti erano uomini, e il modello ha imparato a replicare questa tendenza.

Domande:

1. Quale tipologia di bias sta influenzando il sistema di selezione del personale?
2. Quali sono le **potenziali violazioni legali** che potrebbero derivare dall'uso di questo modello? (Es. discriminazione, violazione delle pari opportunità)
3. Quali azioni correttive potrebbero essere intraprese per rendere il sistema più equo?

10.7.3. Esercizio 3: Analisi di un Caso Reale di Bias Algoritmico

Obiettivo: Riflettere sugli effetti reali del bias nei modelli di machine learning.

Attività:

1. **Ricerca un caso reale** (esempio: COMPAS, bias nei sistemi di riconoscimento facciale, discriminazione nei sistemi di credito).
2. **Descrivi il problema**: Qual era il bias riscontrato nel modello?
3. **Analizza le conseguenze**: Quali impatti ha avuto il bias sulle persone o sulle istituzioni coinvolte?
4. **Valuta le misure adottate**: Sono state prese azioni per correggere il bias? Se sì, quali?

Suggerimento: Se non trovi un caso, puoi analizzare il caso **COMPAS**, un software utilizzato negli USA per prevedere la recidiva criminale, che ha mostrato discriminazioni razziali nelle sue previsioni.

11. Regressione Lineare e Logistica

11.1. Introduzione alla Regressione Lineare

La **regressione lineare** è uno degli algoritmi più semplici ed efficaci per prevedere un valore continuo. Questo modello è alla base di numerosi metodi statistici e di machine learning ed è utilizzato in svariati contesti, dalla finanza alla giurisprudenza (Bishop 2006).

11.1.1. Definizione Matematica

Matematicamente, la regressione lineare è descritta dall'equazione:

$$\hat{y} = \beta_0 + \beta_1 x_1$$

Dove: - $\hat{y}$ è il valore predetto della variabile dipendente. - x_1 è la variabile indipendente. - β_0 è l'intercetta (valore di $\hat{y}$ quando $x_1 = 0$). - β_1 è il coefficiente di regressione che determina l'influenza di x_1 su $\hat{y}$.

L'obiettivo della regressione lineare è trovare i coefficienti che minimizzano l'**errore quadratico medio** (Mean Squared Error, MSE):

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

Questa funzione di errore misura la differenza tra i valori reali e quelli predetti. La regressione lineare utilizza metodi come i **minimi quadrati** o la **discesa del gradiente** per trovare i valori ottimali di β (Hastie, Tibshirani, and Friedman 2009).

11.2. Regressione Lineare Multipla

La **regressione lineare multipla** è un'estensione della regressione lineare semplice che include più variabili indipendenti. È particolarmente utile quando si vogliono modellare relazioni più complesse tra variabili.

11.2.1. Definizione Matematica

La formula della regressione lineare multipla è:

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

Dove: - $\hat{y}$ è la variabile dipendente predetta. - $x_1, x_2, \ldots, x_n$ sono le variabili indipendenti. - β_0 è l'intercetta. - $\beta_1, \beta_2, \ldots, \beta_n$ sono i coefficienti di regressione che indicano il peso di ogni variabile indipendente.

L'obiettivo rimane lo stesso: minimizzare l'errore tra le previsioni e i valori reali. Tuttavia, con più variabili, il modello diventa più complesso e richiede strumenti avanzati per identificare le variabili più rilevanti.

11.2.2. Selezione delle Variabili Più Influenti

Quando si lavora con la **regressione lineare multipla**, una delle sfide principali è determinare quali variabili indipendenti influenzano maggiormente la variabile dipendente. L'uso di troppe variabili può portare a **overfitting**, mentre l'esclusione di variabili importanti può causare **underfitting**.

11.2.2.1. Metodi per la Selezione delle Variabili

Alcuni metodi per selezionare le variabili più influenti includono:

1. **Analisi dei Coefficienti**: Se un coefficiente β_i è vicino a zero, la variabile corrispondente potrebbe avere un impatto trascurabile.
2. **Eliminazione Stepwise (Backward o Forward Selection)**: Tecniche iterative per rimuovere o aggiungere variabili in base alla loro significatività statistica.
3. **LASSO (Least Absolute Shrinkage and Selection Operator)**: Metodo di regressione che impone una penalizzazione sui coefficienti, riducendo a zero quelli meno rilevanti.
4. **Importanza delle Variabili nei Modelli di Alberi**: Anche se si sta usando la regressione, tecniche come le **random forest** possono aiutare a identificare le variabili più influenti.

11.3. Introduzione alla Regressione Logistica

La **regressione logistica** è una tecnica di classificazione utilizzata quando la variabile dipendente può assumere due valori distinti, come "colpevole" o "non colpevole". Questo metodo è ampiamente utilizzato per la valutazione del rischio di recidiva (Kleinberg, Mullainathan, and Raghavan 2018).

11.3.1. Definizione Matematica

La regressione logistica applica una trasformazione sigmoide alla regressione lineare:

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

La funzione sigmoide restituisce valori tra 0 e 1, interpretati come probabilità. Se $\hat{y} > 0.5$, il modello assegna la classe 1, altrimenti assegna la classe 0.

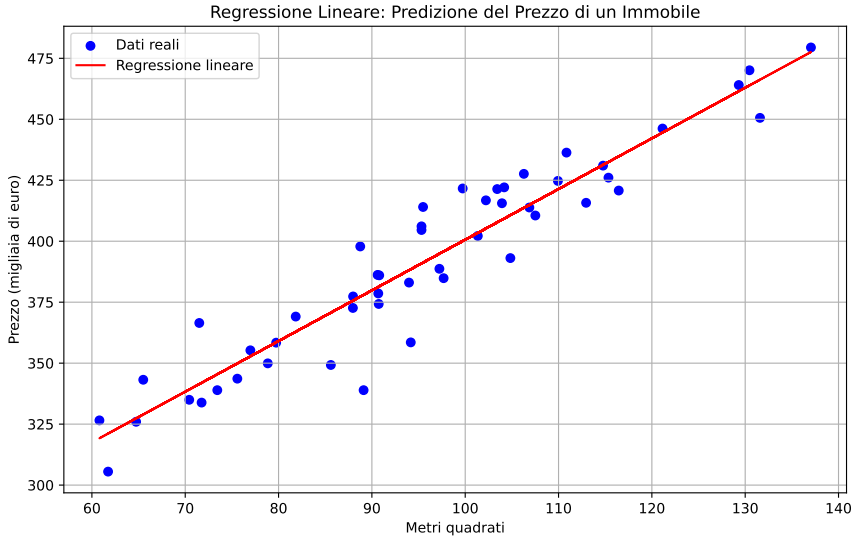
11.4. Connessione con le Reti Neurali

Le reti neurali sono un'estensione della regressione lineare e logistica. Ogni **neurone** in una rete neurale esegue una combinazione lineare delle variabili di input, seguita da una funzione di attivazione, proprio come nella regressione logistica. Comprendere questi modelli è essenziale per lo studio del **deep learning** (Goodfellow, Bengio, and Courville 2016).

11.5. Laboratorio di Python

11.5.1. Esperimento 1: Regressione Lineare (Stima del prezzo di un immobile)

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

# Generazione dati simulati
np.random.seed(42)
X = np.random.normal(100, 20, 50).reshape(-1, 1)
y = 200 + 2 * X.flatten() + np.random.normal(0, 15,
 ↪  X.shape[0])

# Creazione e addestramento del modello
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

# Visualizzazione
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()
```



Questo codice Python implementa un semplice modello di **regressione lineare** per stimare il prezzo di un immobile in base alla sua superficie (metri quadrati). Vediamo passo dopo passo cosa fa il codice.

1. Importazione delle librerie

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

- **numpy** → usato per generare e manipolare dati numerici.
- **matplotlib.pyplot** → usato per creare grafici.
- **sklearn.linear_model.LinearRegression** → implementa un modello di regressione lineare.

2. Generazione dei dati simulati

```
np.random.seed(42)
X = np.random.normal(100, 20, 50).reshape(-1, 1)
y = 200 + 2 * X.flatten() + np.random.normal(0, 15,
 ↪  X.shape[0])
```

- `np.random.seed(42)` → imposta un seme per la generazione casuale, garantendo che i risultati siano riproducibili.
- `np.random.normal(100, 20, 50)` → genera 50 numeri casuali con media 100 e deviazione standard 20 (rappresentano i metri quadrati delle case).
- `reshape(-1, 1)` → trasforma l'array in una matrice con una colonna e 50 righe (richiesto da `sklearn`).
- `y = 200 + 2 * X.flatten() + np.random.normal(0, 15, X.shape[0])` → crea i prezzi delle case seguendo una relazione lineare:
  - `200` è l'intercetta (prezzo base).
  - `2 * X` significa che il prezzo aumenta di 2 unità per ogni metro quadrato in più.
  - `np.random.normal(0, 15, X.shape[0])` aggiunge rumore casuale ai prezzi per simulare dati reali.

3. Creazione e addestramento del modello di regressione lineare

```
model = LinearRegression()
model.fit(X, y)
```

- **LinearRegression()** → crea un modello di regressione lineare.
- **model.fit(X, y)** → addestra il modello usando i dati (`X` come input e y come output).

4. Predizione dei valori

```
y_pred = model.predict(X)
```

- `model.predict(X)` → usa il modello addestrato per calcolare il prezzo stimato delle case.

5. Visualizzazione dei risultati

```
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()
```

- **plt.scatter(X, y, color='blue', label='Dati reali')** → disegna un grafico a dispersione con i dati reali (metri quadrati vs prezzo).
- **plt.plot(X, y_pred, color='red', label='Regressione lineare')** → disegna la linea di regressione che approssima i dati.
- **Aggiunge etichette, legenda e griglia** per migliorare la leggibilità.

Cosa rappresenta il grafico?

- I **punti blu** sono i dati reali (metri quadrati vs prezzo dell'immobile).

- La **linea rossa** rappresenta la retta della regressione lineare, ovvero la previsione del modello.
- L'obiettivo è trovare la linea che meglio approssima i dati per poter stimare il prezzo di una casa conoscendone solo i metri quadrati.

Conclusione: Il modello impara una relazione lineare tra la superficie dell'immobile e il prezzo, permettendo di stimare il valore di case future basandosi sui metri quadrati!

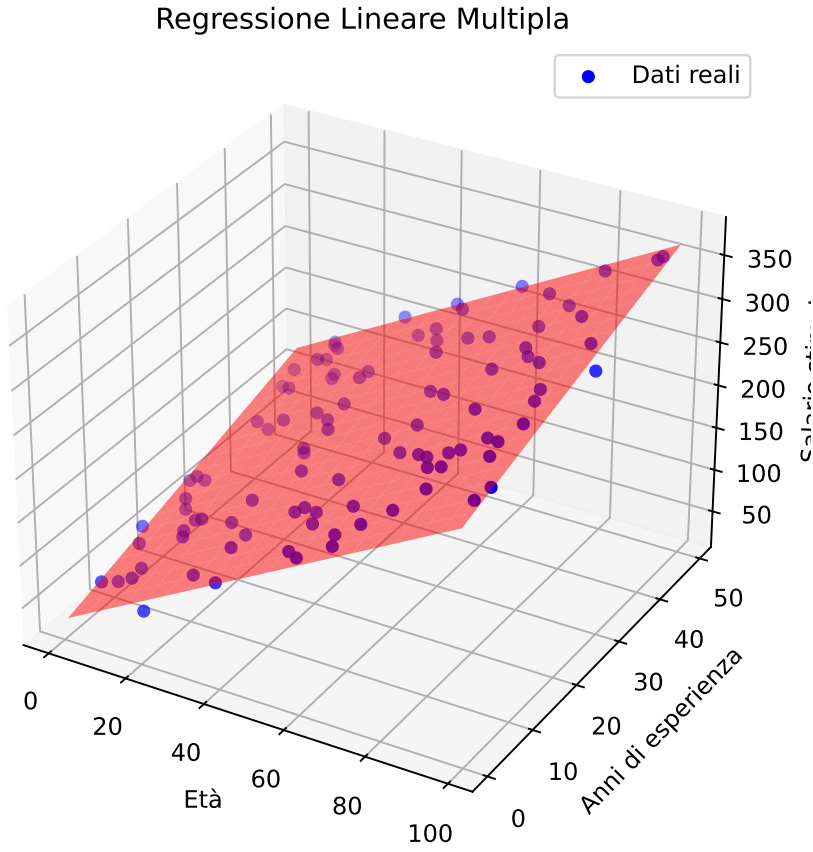
11.5.2. Esperimento 2: Regressione Lineare Multipla

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.linear_model import LinearRegression

# Creazione di un dataset simulato
np.random.seed(42)
n = 100
X1 = np.random.rand(n) * 100  # Variabile
 ↪  indipendente 1 (es. età)
X2 = np.random.rand(n) * 50   # Variabile
 ↪  indipendente 2 (es. anni di esperienza)
y = 30 + 2.5 * X1 + 1.8 * X2 + np.random.normal(0,
 ↪  10, n)  # Output con rumore

# Creazione del modello di regressione lineare
 ↪  multipla
```

```
X = np.column_stack((X1, X2))
model = LinearRegression()
model.fit(X, y)

# Predizione dei valori
y_pred = model.predict(X)

# Visualizzazione dei coefficienti
print("Coefficienti:", model.coef_)
print("Intercetta:", model.intercept_)

# Creazione del grafico 3D per visualizzare i dati e
 ↪  il piano di regressione
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot dei dati reali
ax.scatter(X1, X2, y, color='blue', label='Dati
 ↪  reali')

# Creazione della superficie di regressione
X1_grid, X2_grid = np.meshgrid(np.linspace(X1.min(),
 ↪  X1.max(), 20),
                               np.linspace(X2.min(),
 ↪  X2.max(), 20))
y_grid = (model.intercept_ + model.coef_[0] *
 ↪  X1_grid + model.coef_[1] * X2_grid)

ax.plot_surface(X1_grid, X2_grid, y_grid,
 ↪  color='red', alpha=0.5)

# Etichette degli assi
ax.set_xlabel("Età")
```

```
ax.set_ylabel("Anni di esperienza")
ax.set_zlabel("Salario stimato")
ax.set_title("Regressione Lineare Multipla")
plt.legend()
plt.show()
```

```
Coefficienti: [2.46582747 1.94386228]
Intercetta: 29.106100364500747
```



Questo codice Python implementa un semplice modello di **regressione lineare multipla** per stimare il salario di un impiegato in base all' età e agli anni di esperienza dell'impiegato. Vediamo passo dopo passo cosa fa il codice.

1 Generazione dei dati Il codice genera un dataset simulato con **100 osservazioni** e **due variabili indipendenti**:

- **X1**: età della persona (tra 0 e 100 anni).
- **X2**: anni di esperienza lavorativa (tra 0 e 50 anni).
- **y**: salario stimato in base a un modello lineare con una certa variabilità casuale (rumore).

La relazione tra le variabili è definita dalla formula:

$$y = 30 + 2.5 \times X1 + 1.8 \times X2 + \text{rumore casuale}$$

- Il valore **30** è l'intercetta (valore base del salario).
- **2.5** è il peso dell'età (ogni anno di età in più aumenta il salario di 2.5 unità).
- **1.8** è il peso degli anni di esperienza (ogni anno di esperienza in più aumenta il salario di 1.8 unità).
- Il **rumore** introduce variazioni casuali per simulare dati realistici.

2 Creazione del modello Dopo aver organizzato i dati in una matrice `X`, il codice utilizza `LinearRegression()` di **Scikit-Learn** per addestrare il modello:

```
model.fit(X, y)
```

Questo calcola i coefficienti che meglio approssimano la relazione tra le variabili.

3 Predizione e interpretazione dei coefficienti Dopo l'addestramento, vengono stampati i coefficienti stimati:

```
print("Coefficienti:", model.coef_)
print("Intercetta:", model.intercept_)
```

Questi valori ci dicono quanto l'età e l'esperienza influenzano il salario.

4 Visualizzazione grafica - Il **grafico 3D** mostra i punti blu (dati reali) e il **piano rosso** che rappresenta la **superficie di regressione**. - Il modello cerca di **minimizzare la distanza** tra il piano e i punti reali.

In questo epserimento abbiamo visto come applicare la **regressione lineare multipla** per prevedere un valore (salario) in base a più variabili (età ed esperienza).

11.5.3. Esperimento 3: Regressione Logistica

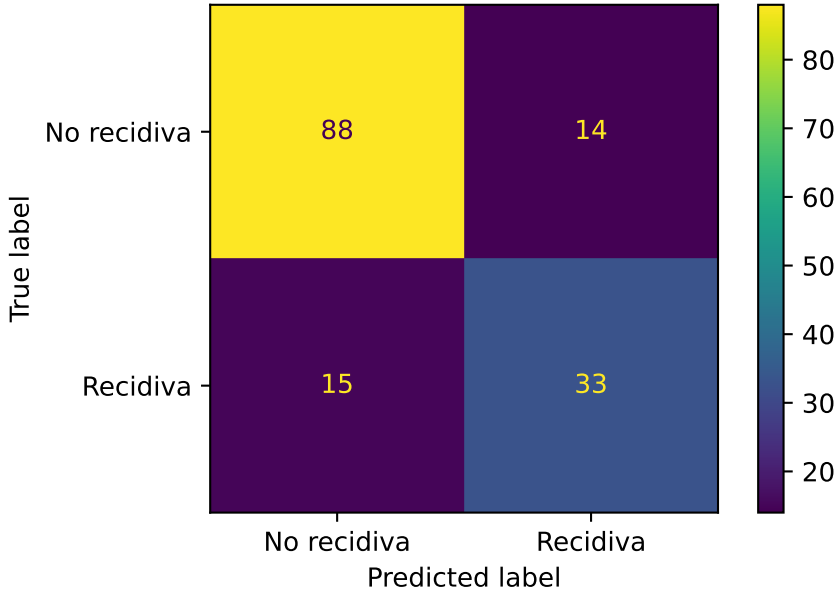
In questo esperimento simuliamo un'applicazione della **regressione logistica** in ambito penale, con l'obiettivo di prevedere la **probabilità di recidiva** di un imputato sulla base di:

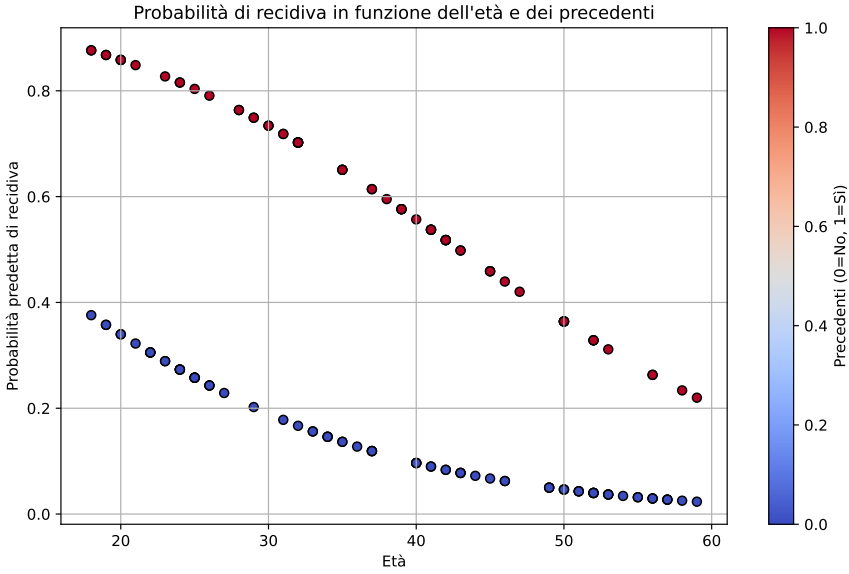
- **Età**,
- **Presenza di precedenti penali**.

```
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 (
    confusion_matrix, ConfusionMatrixDisplay,
↪   classification_report
```

```
)

# 1. Simulazione dati: rischio di recidiva in
 ↪  funzione di età e precedenti
np.random.seed(42)
n = 500
eta = np.random.randint(18, 60, n)  # età tra 18 e
 ↪  60
precedenti = np.random.binomial(1, 0.4, n)  # 0 =
 ↪  nessun precedente, 1 = precedenti

# Funzione probabilità: maggiore con precedenti e
 ↪  minore età
p_recidiva = 1 / (1 + np.exp(-(-4 + 0.08 * (60 -
 ↪  eta) + 2.5 * precedenti)))
y = np.random.binomial(1, p_recidiva)

# Preparazione dataset
X = np.column_stack((eta, precedenti))
X_train, X_test, y_train, y_test =
 ↪  train_test_split(X, y, test_size=0.3,
 ↪  random_state=42)

# 2. Regressione Logistica
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]

# 3. Valutazione
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred,
 ↪  output_dict=True)
```

```
# 4. Visualizzazione
ConfusionMatrixDisplay(confusion_matrix=cm,
 ↪  display_labels=["No recidiva",
 ↪  "Recidiva"]).plot()

# Decision boundary
plt.figure(figsize=(10, 6))
plt.scatter(X_test[:, 0], y_prob, c=X_test[:, 1],
 ↪  cmap='coolwarm', edgecolor='k')
plt.xlabel("Età")
plt.ylabel("Probabilità predetta di recidiva")
plt.title("Probabilità di recidiva in funzione
 ↪  dell'età e dei precedenti")
plt.colorbar(label="Precedenti (0=No, 1=Sì)")
plt.grid(True)
plt.show()

(report, model.coef_, model.intercept_)
```






```
({'0': {'precision': 0.8543689320388349,
```

```
   'recall': 0.8627450980392157,
   'f1-score': 0.8585365853658536,
   'support': 102.0},
  '1': {'precision': 0.7021276595744681,
   'recall': 0.6875,
   'f1-score': 0.6947368421052632,
   'support': 48.0},
  'accuracy': 0.8066666666666666,
  'macro avg': {'precision': 0.7782482958066514,
   'recall': 0.7751225490196079,
   'f1-score': 0.7766367137355584,
   'support': 150.0},
  'weighted avg': {'precision': 0.8056517248502376,
   'recall': 0.8066666666666666,
   'f1-score': 0.8061206675224647,
   'support': 150.0}},
 array([[-0.07867108,  2.46644213]]),
 array([0.90914022]))
```

Il codice Python implementa un semplice modello di **regressione logistica** per prevedere la probabilità di recidiva di un imputato sulla base di età e presenza di precedenti penali. Vediamo passo dopo passo cosa fa il codice.

Abbiamo generato un dataset fittizio di 500 imputati, in cui:

- L'età è compresa tra 18 e 60 anni.
- Il 40% ha precedenti penali.
- La probabilità di recidiva è più alta per:
  - imputati con precedenti,
  - imputati più giovani.

Questa probabilità è calcolata con una funzione logistica:

$$p = \frac{1}{1 + \exp(-(-4 + 0.08 \cdot (60 - \text{età}) + 2.5 \cdot \text{precedenti}))}$$

I valori di y (recidiva sì/no) sono stati estratti in base a questa probabilità.

Abbiamo addestrato una **regressione logistica** sui dati, divisi in train e test set.

Il modello ha appreso i seguenti coefficienti:

- **Età**: −0.078 → l'aumento dell'età riduce la probabilità di recidiva.
- **Precedenti**: +2.47 → avere precedenti penali aumenta significativamente la probabilità.

Intercetta: +0.91

Per quanto concerne l'accuratezza del modello, abbiamo ottenuto:

Accuratezza complessiva: 80.6% **Precisione per la classe "recidiva"**: 70.2% **Richiamo (recall) per "recidiva"**: 68.8%

La matrice di confusione mostra una buona capacità del modello nel distinguere tra recidiva e non recidiva.

Nel secondo grafico, ogni punto rappresenta un imputato del test set:

- **X**: Età
- **Y**: Probabilità predetta di recidiva
- **Colore**: Presenza di precedenti (blu = no, rosso = sì)

La curva mostra come la probabilità predetta diminuisce con l'età e aumenta sensibilmente in presenza di precedenti penali.

11.6. Esercizi

11.6.1. Esercizio 1: Analisi delle Variabili

Quali tecniche si possono usare per selezionare le variabili più influenti in una regressione lineare multipla?

11.6.2. Esercizio 2: Interpretazione della Regressione Logistica

Se un coefficiente della regressione logistica ha valore negativo, cosa implica per la probabilità predetta?

11.6.3. Esercizio 3: Effetti di un Dataset Sbilanciato

Un dataset con il 90% dei casi appartenenti a una sola classe può influenzare la regressione logistica? Come si può correggere questo problema?

12. Percetptrone

12.1. Introduzione

Il **percettrone** è un modello semplificato del funzionamento di una cellula nervosa, proposto da Frank Rosenblatt nel 1958 (Rosenblatt 1958). Rappresenta la base su cui si sono costruite le moderne reti neurali ed è in grado di risolvere problemi di classificazione **linearmente separabili**. La sua semplicità lo rende un ottimo punto di partenza per comprendere i principi fondamentali del machine learning.

12.2. Struttura e Funzionamento

Il percettrone prende in input un vettore di caratteristiche numeriche (x = (x_1, x_2, …, x_n)) e calcola una **somma pesata**:

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

Dove: - w_i sono i pesi, - $b è il bias.

Il risultato z viene trasformato da una **funzione di attivazione** che produce l'output del modello:

$$\hat{y} = f(z)$$

Nel percettrone classico, la funzione f è la **funzione a soglia**:

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

12.3. Addestramento del Percettrone

Il modello viene addestrato con l'**algoritmo di aggiornamento del percettrone**. Per ogni errore di classificazione, i pesi vengono aggiornati come segue:

$$w_i \leftarrow w_i + \eta \cdot (y - \hat{y}) \cdot x_i$$

$$b \leftarrow b + \eta \cdot (y - \hat{y})$$

Dove: - η è il **tasso di apprendimento** (learning rate), - y è il valore reale, - $\hat{y}$ è il valore predetto.

Questo processo continua finché il modello non commette più errori (o si raggiunge un numero massimo di iterazioni).

12.4. Funzioni di Attivazione

Oltre alla soglia, esistono molte altre funzioni di attivazione usate nei neuroni artificiali:

12.4.1. Funzione Sigmoide

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

- Output continuo tra 0 e 1.
- Usata nei modelli probabilistici, come la regressione logistica.

12.4.2. Funzione Tanh

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

- Output tra -1 e 1.
- Più centrata rispetto alla sigmoide.

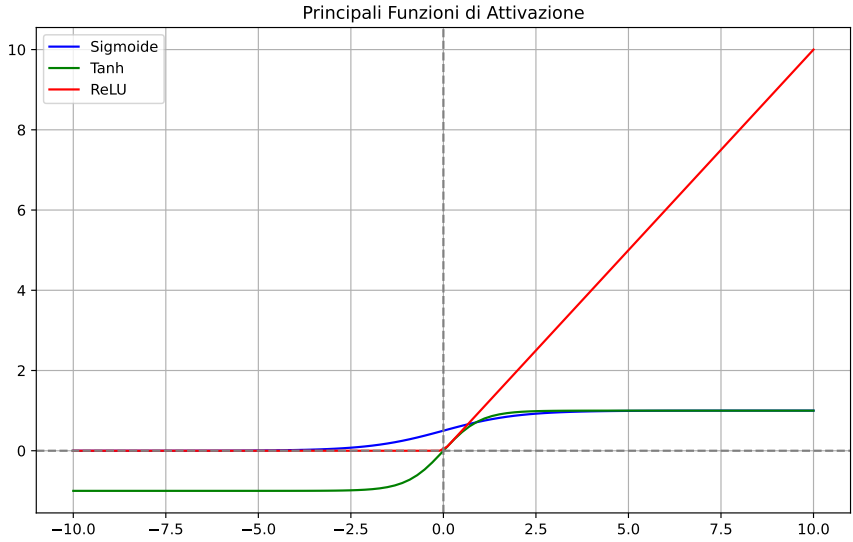
12.4.3. Funzione ReLU (Rectified Linear Unit)

$$f(z) = \max(0, z)$$

- Attiva solo valori positivi.
- Molto usata nelle reti neurali profonde (Goodfellow, Bengio, and Courville 2016).

L'andamento delle funzioni di attivazioni viste è mostrato nel seguente grafico.

```
import numpy as np
import matplotlib.pyplot as plt

z = np.linspace(-10, 10, 100)
sigmoid = 1 / (1 + np.exp(-z))
tanh = np.tanh(z)
relu = np.maximum(0, z)

plt.figure(figsize=(10, 6))
plt.plot(z, sigmoid, label='Sigmoide', color='blue')
plt.plot(z, tanh, label='Tanh', color='green')
plt.plot(z, relu, label='ReLU', color='red')
plt.axhline(0, color='gray', linestyle='--')
plt.axvline(0, color='gray', linestyle='--')
plt.title("Principali Funzioni di Attivazione")
plt.legend()
plt.grid(True)
plt.show()
```



12.5. Limitazioni

Il percettrone può risolvere solo problemi **linearmente separabili**, cioè in cui una retta (o un iperpiano) può separare le classi. Per problemi più complessi, si usano **reti neurali multilivello (MLP)**, in grado di apprendere anche confini non lineari (Haykin 2009).

12.6. Laboratorio di Python

12.6.1. Esperimento 1: Implementazione di un perceptrone

In questo esperimento sviluppiamo una implementazione del perceptrone senza fare uso di librerie esterne e lo useremo in un

problema di classificazione con classi linearmente separabili. Le sole librerie usata sono **random** per la generazione di numeri casuali per simulare i dati e **matplotlib** per graficare i risultati. I dati simulati sono costruti per essere linearmente separabili.

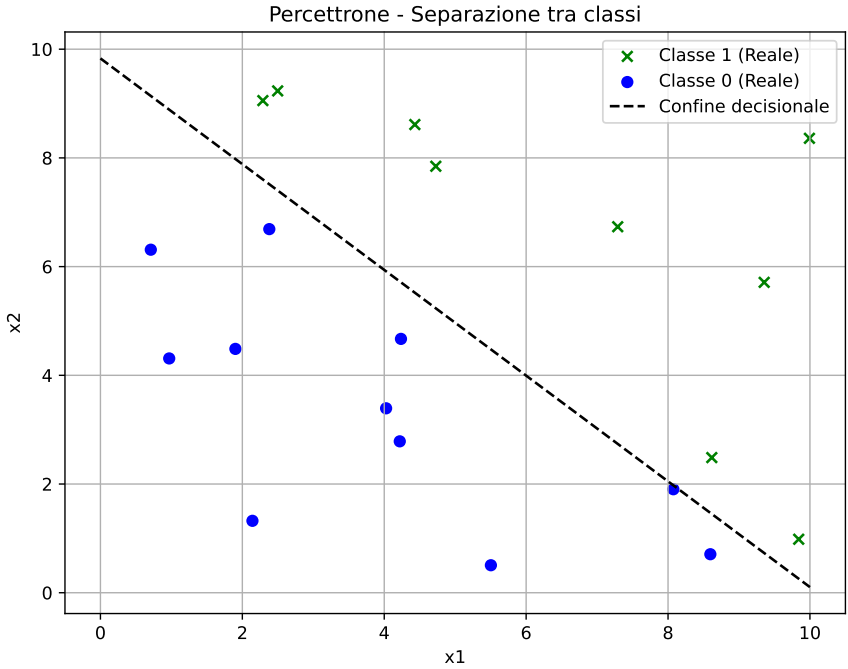
```
import random
import matplotlib.pyplot as plt

# ----------------------
# 1. Generazione del dataset
# ----------------------

random.seed(42)
n_samples = 100
X = []
y = []

for _ in range(n_samples):
    x1 = random.uniform(0, 10)
    x2 = random.uniform(0, 10)
    label = 1 if x1 + x2 > 10 else 0  # Separazione
↪   lineare lungo la diagonale x1 + x2 = 10
    X.append([x1, x2])
    y.append(label)

# Suddividiamo manualmente in training e test set
split_index = int(0.8 * n_samples)
X_train = X[:split_index]
y_train = y[:split_index]
X_test = X[split_index:]
y_test = y[split_index:]

# ----------------------
# 2. Funzioni del Percettrone
```

```
# ----------------------

def step_function(z):
    return 1 if z >= 0 else 0

def predict(x, weights, bias):
    z = sum(w * xi for w, xi in zip(weights, x)) +
 ↪  bias
    return step_function(z)

def train_perceptrone(X_train, y_train,
 ↪  learning_rate=0.1, epochs=10):
    n_features = len(X_train[0])
    weights = [0.0 for _ in range(n_features)]
    bias = 0.0

    for epoch in range(epochs):
        for x, label in zip(X_train, y_train):
            y_pred = predict(x, weights, bias)
            error = label - y_pred
            # Aggiornamento pesi e bias
            for i in range(n_features):
                weights[i] += learning_rate * error
 ↪  * x[i]
            bias += learning_rate * error
    return weights, bias

# ----------------------
# 3. Addestramento del modello
# ----------------------

weights, bias = train_perceptrone(X_train, y_train,
 ↪  learning_rate=0.1, epochs=30)
```

```
# -----------------------
# 4. Valutazione e Matrice di Confusione
# -----------------------

TP = FP = TN = FN = 0
y_pred_test = []

for x, label in zip(X_test, y_test):
    y_hat = predict(x, weights, bias)
    y_pred_test.append(y_hat)
    if y_hat == 1 and label == 1:
        TP += 1
    elif y_hat == 1 and label == 0:
        FP += 1
    elif y_hat == 0 and label == 0:
        TN += 1
    elif y_hat == 0 and label == 1:
        FN += 1

print("Matrice di Confusione (senza librerie
 ↪  esterne):")
print(f"TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}")
accuracy = (TP + TN) / len(y_test)
print(f"Accuratezza: {accuracy:.2f}")

# -----------------------
# 5. Visualizzazione
# -----------------------

x1_pos = [X_test[i][0] for i in range(len(y_test))
 ↪  if y_test[i] == 1]
x2_pos = [X_test[i][1] for i in range(len(y_test))
 ↪  if y_test[i] == 1]
```

```
x1_neg = [X_test[i][0] for i in range(len(y_test))
 ↪  if y_test[i] == 0]
x2_neg = [X_test[i][1] for i in range(len(y_test))
 ↪  if y_test[i] == 0]

plt.figure(figsize=(8, 6))
plt.scatter(x1_pos, x2_pos, c='green', label='Classe
 ↪  1 (Reale)', marker='x')
plt.scatter(x1_neg, x2_neg, c='blue', label='Classe
 ↪  0 (Reale)', marker='o')

# Linea di decisione: w1*x1 + w2*x2 + b = 0 => x2 =
 ↪  -(w1*x1 + b)/w2
x1_vals = [i for i in range(0, 11)]
if weights[1] != 0:
    x2_vals = [-(weights[0]*x + bias)/weights[1] for
 ↪  x in x1_vals]
    plt.plot(x1_vals, x2_vals, '--', color='black',
 ↪  label='Confine decisionale')

plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Percettrone - Separazione tra classi")
plt.legend()
plt.grid(True)
plt.show()
```

```
Matrice di Confusione (senza librerie esterne):
TP: 9, FP: 0, TN: 11, FN: 0
Accuratezza: 1.00
```



Il codice Python usato per l'implementazione del perceptrone e per il grafico dei risultati segue la seguente logica:

1. **Generazione dati** Simuliamo 100 punti 2D. I punti per cui $x_1 + x_2 > 10$ sono etichettati come **classe 1**, altrimenti **classe 0**.

2. **Implementazione del percettrone**

- `step_function(z)`: restituisce 1 se la somma è maggiore o uguale a zero, altrimenti 0.
- `predict(x, weights, bias)`: calcola il valore di z e applica la funzione di attivazione.
- `train_perceptrone()`: implementa l'algoritmo di apprendimento con aggiornamento dei pesi.

3. **Addestramento** Il modello viene addestrato per 20 epoche (ripetizioni del dataset) su 80% dei dati.

4. **Matrice di confusione** Calcolo dei 4 elementi della matrice di confusione:

   - **TP**: (True Positive = Vero Positivo) predetto 1 e vero 1
   - **TN**: (True Negative = Vero Negativo) predetto 0 e vero 0
   - **FP**: (False Positive = Falso Positivo) predetto 1 e vero 0 (falso allarme)
   - **FN**: (False Negative = Falso Negativo) predetto 0 e vero 1 (falso negativo)

5. **Visualizzazione** Il grafico mostra:

   - I punti veri (blu per classe 0, verde per classe 1).
   - Il **confine decisionale** calcolato con la formula della retta.

Risultato atteso

Con dati così semplici e ben separabili, il percettrone dovrebbe ottenere una **accuratezza alta (quasi 100%)**.

12.6.2. Esperimento 2: il perceptrone bel caso di classi concentriche

Il codice Python che segue implementa un semplice modello di classificatore a **perceptrone** usando la sola libreria random per la generazione di numeri casuali per simulare i dati e matplotlib per graficare i risultati. I dati simulati sono distribuiti su due corone circolari concentriche e quindi non linearmente separabili.

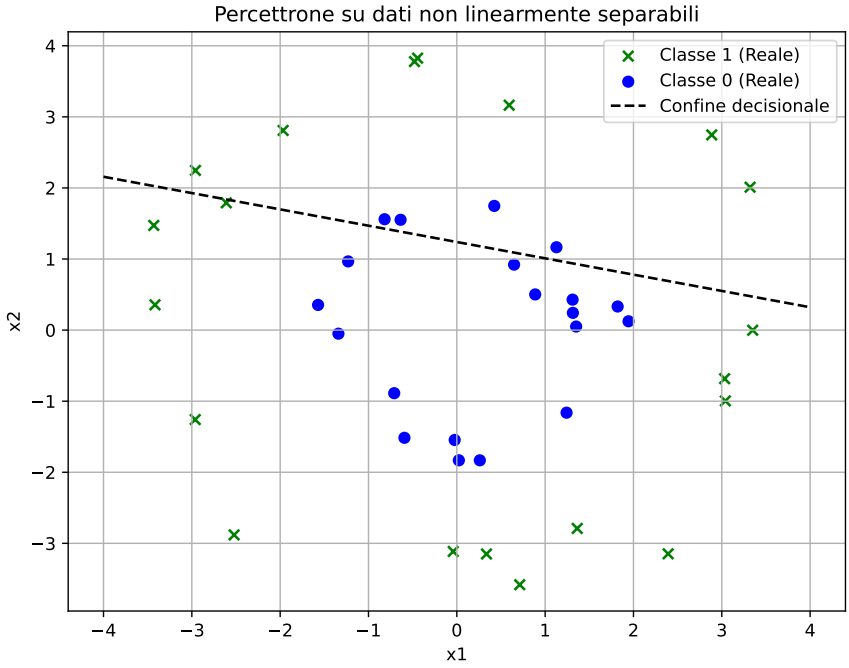
```
import math
import random
import matplotlib.pyplot as plt

# ------------------------------
# 1. Generazione del dataset circolare
# ------------------------------

def genera_cerchi(n_samples):
    X = []
    y = []
    for _ in range(n_samples):
        r = random.uniform(0, 1)
        angle = random.uniform(0, 2 * math.pi)
        if r < 0.5:
            radius = random.uniform(1, 2)
            label = 0
        else:
            radius = random.uniform(3, 4)
            label = 1
        x1 = radius * math.cos(angle)
        x2 = radius * math.sin(angle)
        X.append([x1, x2])
        y.append(label)
    return X, y

random.seed(42)
X, y = genera_cerchi(200)

# Suddividiamo in training e test set
split_index = int(0.8 * len(X))
X_train = X[:split_index]
y_train = y[:split_index]
```

```
X_test = X[split_index:]
y_test = y[split_index:]

# ------------------------------
# 2. Percettrone da zero
# ------------------------------

def step_function(z):
    return 1 if z >= 0 else 0

def predict(x, weights, bias):
    z = sum(w * xi for w, xi in zip(weights, x)) +
 ↪  bias
    return step_function(z)

def train_perceptrone(X, y, learning_rate=0.1,
 ↪  epochs=20):
    n_features = len(X[0])
    weights = [0.0 for _ in range(n_features)]
    bias = 0.0
    for epoch in range(epochs):
        for xi, yi in zip(X, y):
            y_pred = predict(xi, weights, bias)
            error = yi - y_pred
            for i in range(n_features):
                weights[i] += learning_rate * error
 ↪  * xi[i]
            bias += learning_rate * error
    return weights, bias

# Addestramento
weights, bias = train_perceptrone(X_train, y_train)
```

```
# ------------------------------
# 3. Valutazione
# ------------------------------

TP = FP = TN = FN = 0
y_pred_test = []

for x, label in zip(X_test, y_test):
    y_hat = predict(x, weights, bias)
    y_pred_test.append(y_hat)
    if y_hat == 1 and label == 1:
        TP += 1
    elif y_hat == 1 and label == 0:
        FP += 1
    elif y_hat == 0 and label == 0:
        TN += 1
    elif y_hat == 0 and label == 1:
        FN += 1

print("Matrice di confusione:")
print(f"TP: {TP}, FP: {FP}, TN: {TN}, FN: {FN}")
accuracy = (TP + TN) / len(y_test)
print(f"Accuratezza: {accuracy:.2f}")

# ------------------------------
# 4. Visualizzazione
# ------------------------------

# Colori reali
x1_pos = [X_test[i][0] for i in range(len(y_test))
 ↪  if y_test[i] == 1]
x2_pos = [X_test[i][1] for i in range(len(y_test))
 ↪  if y_test[i] == 1]
```

```
x1_neg = [X_test[i][0] for i in range(len(y_test))
 ↪  if y_test[i] == 0]
x2_neg = [X_test[i][1] for i in range(len(y_test))
 ↪  if y_test[i] == 0]

plt.figure(figsize=(8, 6))
plt.scatter(x1_pos, x2_pos, color='green',
 ↪  label='Classe 1 (Reale)', marker='x')
plt.scatter(x1_neg, x2_neg, color='blue',
 ↪  label='Classe 0 (Reale)', marker='o')

# Linea di decisione (valida solo per separazione
 ↪  lineare)
x_vals = [i / 10.0 for i in range(-40, 41)]
if weights[1] != 0:
    y_vals = [-(weights[0]*x + bias)/weights[1] for
 ↪  x in x_vals]
    plt.plot(x_vals, y_vals, '--', color='black',
 ↪  label='Confine decisionale')

plt.title("Percettrone su dati non linearmente
 ↪  separabili")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.grid(True)
plt.show()
```

```
Matrice di confusione:
TP: 13, FP: 16, TN: 4, FN: 7
Accuratezza: 0.42
```



L'esito dell'esperimento mostra chiaramente ciò che già sapevamo bene: Il **percettrone può apprendere solo confini lineari**. In questo caso:

- I dati sono disposti in **cerchi concentrici**, cioè non possono essere separati con una retta.
- Il percettrone cerca una retta per dividere i dati… ma non riesce e commette molti errori.
- L'**accuratezza è bassa** (minore del 50%).
- Il grafico mostra che la **linea di decisione** non riesce a separare correttamente le due classi.

In casi come questo servono modelli di reti neurali più avanzati come le **reti neurali multistrato (MLP)** .

12.7. Esercizi

12.7.1. Esercizio 1: Comprendere la classificazione lineare

Spiega con parole tue cosa significa che un problema è *linearmente separabile*. Fai un esempio nel contesto giuridico in cui questo tipo di separazione potrebbe essere realistico.

12.7.2. Esercizio 2: Aggiornamento dei pesi

Dati i seguenti valori:

- input $x = [2, -1]$
- pesi iniziali $w = [0.5, 0.5]$
- bias $b = 0.2$
- apprendimento $\eta = 0.1$
- output vero $y = 1$
- output predetto $\hat{y} = 0$

Calcola i nuovi valori di w e b dopo un singolo aggiornamento.

12.7.3. Esercizio 3: Vantaggi e limiti del modello

Quali sono i principali vantaggi e limiti dell'utilizzo del percettrone in ambito giuridico? Riporta almeno un esempio per ciascun caso.

13. Deep learning

13.1. Introduzione

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(Goodfellow, Bengio, and Courville 2016). 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.

13.2. 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(Haykin 2009). Questa tecnica è fondamentale non solo per le MLP, ma anche per le reti neurali più complesse utilizzate nel deep learning.

13.2.1. 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.

13.2.2. Funzioni di Attivazione

Le funzioni di attivazione sono cruciali per introdurre la non linearità nelle reti neurali, permettendo al modello multistrato di apprendere pattern complessi. Come visto in Section 12.6.2 un singolo perceptrone non riesce a classificare dati che non sono lineramente separabili. In questi casi occorre usare una rete multistrato con funzioni di attivazione non lineari nei neuroni artificiali degli strati intermedi. La non linearità è fondamentale: se si usassero solo funzioni di attivazione lineari, l'intera rete si comporterebbe come un singolo strato lineare. In altre parole, l'uscita sarebbe ancora una combinazione lineare degli ingressi, rendendo la rete incapace di apprendere relazioni complesse e limitandola a definire solo classificatori lineari (anche se in modo più articolato). Al contrario, la combinazione di funzioni non lineari dà la possibilità di approssimare funzioni complesse e definire curve di separazione tra classi (o superfici in più dimensioni) arbitrariamente elaborate. Si veda Section 13.9.2 per una soluzione al problema di classificare dati distribuiti in classi concentriche con reti multistrato. 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.

13.2.3. 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.

13.2.4. 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:

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

13.3. Algoritmo di Backpropagation

Il **backpropagation** è l'algoritmo chiave che permette l'addestramento delle reti neurali multistrato(Rumelhart, Hinton, and Williams 1986). Funziona in due fasi principali:

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 l'ottimizzazione tramite **discesa del gradiente**, minimizzando così la funzione di perdita.

Il processo è iterativo e viene ripetuto molte volte fino a quando il modello non raggiunge un buon livello di accuratezza.

13.3.1. Epoche e apprendimento iterativo

Un'**epoca** (epoch) rappresenta un ciclo completo attraverso l'intero dataset di addestramento. Durante un'epoca, tutti gli esempi presenti nel dataset vengono utilizzati per aggiornare i pesi del modello. Tuttavia, per migliorare l'efficienza e la stabilità dell'apprendimento, il dataset viene tipicamente suddiviso in **batch**.

13.3.2. Batch di dati di addestramento

Un **batch** è un sottoinsieme del dataset utilizzato per aggiornare i pesi una volta. Questo approccio prende il nome di **addestramento mini-batch**. Esistono tre strategie principali:

- **Batch learning**: tutto il dataset viene usato in un solo passo per ogni aggiornamento dei pesi. È computazionalmente intenso.
- **Stochastic Gradient Descent (SGD)**: ogni esempio di addestramento aggiorna i pesi immediatamente. È rumoroso ma può favorire la convergenza a un minimo globale.
- **Mini-batch Gradient Descent**: è il metodo più usato. I dati sono suddivisi in piccoli gruppi (batch), e ciascun batch viene utilizzato per calcolare un aggiornamento dei pesi. Questo equilibrio consente stabilità ed efficienza.

Ad esempio, se abbiamo 10.000 esempi e un batch size di 100, avremo 100 batch per ogni epoca. Durante ogni epoca, il modello vedrà l'intero dataset, ma aggiornando i pesi 100 volte invece di una sola.

Organizzare l'apprendimento su più epoche permette alla rete di apprendere progressivamente dai dati. Durante l'addestramento, è comune monitorare l'errore sul **set di validazione** per evitare l'**overfitting** e applicare tecniche come l'**early stopping**, che interrompe l'addestramento se le prestazioni peggiorano su tale set.

In sintesi, l'algoritmo di backpropagation con l'organizzazione in epoche e batch costituisce il cuore del processo di apprendimento nelle reti neurali profonde, permettendo l'ottimizzazione dei pesi in modo efficiente e scalabile.

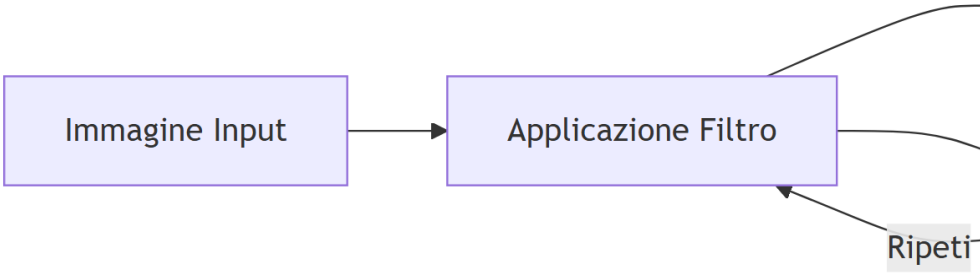
13.4. Architetture di Reti Neurali Profonde

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.

13.4.1. 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 (LeCun, Bengio, and Hinton 2015). 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.



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, l'individuazione di oggetti, e la segmentazione semantica, superando spesso le capacità umane in questi domini.

13.4.2. 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(Karpathy 2015).

13.4.3. Reti Generative Adversariali (GAN)

Le GAN rappresentano un'architettura avanzata che contrappone due reti neurali, una generativa e una discriminativa, in un meccanismo competitivo (o framework competitivo)(Goodfellow

et al. 2014). 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.

13.5. 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.

13.6. 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.

13.7. 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.

13.8. 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.

13.9. Laboratorio Python

13.9.1. Esperimento 1: Rappresentazione grafiche di reti neurali multistrato

In questo esperimento vogliamo programmare una funzione in grado di realizzare una rappresentazione grafica di una rete neurale. A tale scopo adotteremo la libreria Python **networkx**. La funzione draw_mlp riceve in ingresso la descrizione della rete multistrato in termini di numero di neuroni di ingresso, numero di strati e numero di neuroni per ogni strato e numero di neuroni di uscita. Il risultato della funzione è il disegno del grafo della rete MLP.

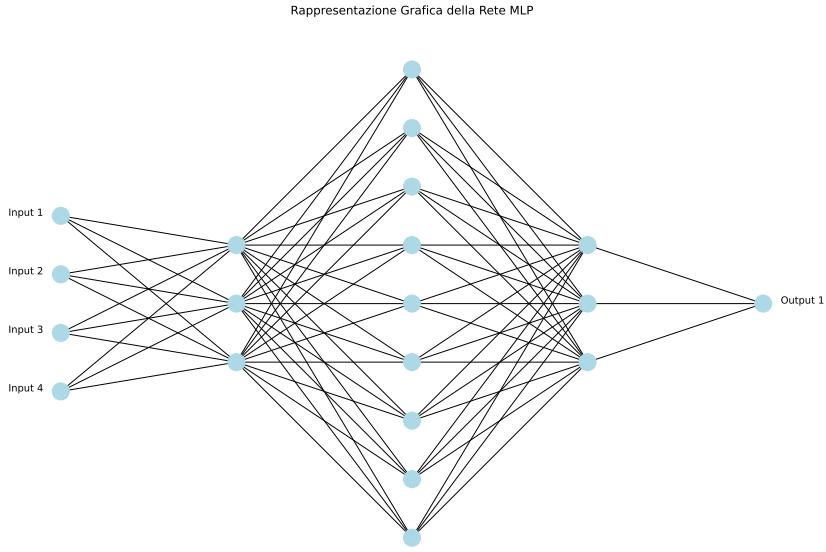
```
import matplotlib.pyplot as plt
import networkx as nx

# 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

    # 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()
```

Applichiamo la funzione draw_mlp() al caso di una rete con 4 ingressi 3 strati nascosti da 3, 9 e 3 neuroni rispettivamente e 1 neurone di uscita:

```
# Parametri della rete MLP utilizzata nell'esempio
hidden_layers = (3,9, 3)  # Due strati nascosti con
 ↪  10 e 5 neuroni rispettivamente
input_size = 4  # Due 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)
```



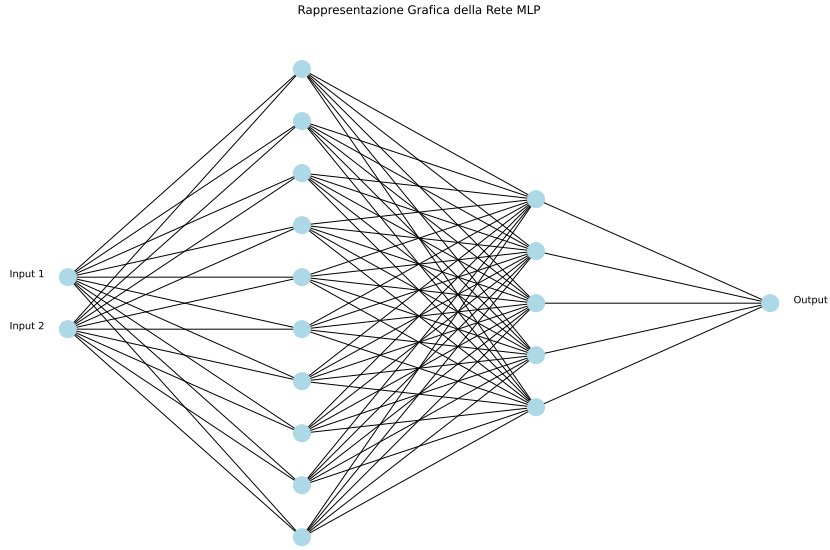
13.9.2. Esperimento 2: Rete MLP applicata al caso di classi concentriche

In questo esperimento vogliamo applicare una rete multistrato al problema della classificazione binaria nel caso di un dataset bidimensionale composto da due classi concentrichele La rete è composta da:

- **Strato di input**: Due ingressi, ciascuno corrispondente a una delle caratteristiche del dataset (x1,x2).
- **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 grazie alla funzione di attivazione non lineare "relu" adottata. 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.
- **Strato di output**: Un singolo neurone di output, utilizzato per la classificazione binaria (classe 0, class 1).

Usando la funzione introdotta nell'esempio 1 possiamo disegnare la rete MLP che vogliamo adottare per risolvere il problem di classificazione nel caso di classi concentriche.

```
# Parametri della rete MLP utilizzata nell'esempio
hidden_layers = (10, 5)  # Due strati nascosti con
 ↪  10 e 5 neuroni rispettivamente
input_size = 2  # Due 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)
```

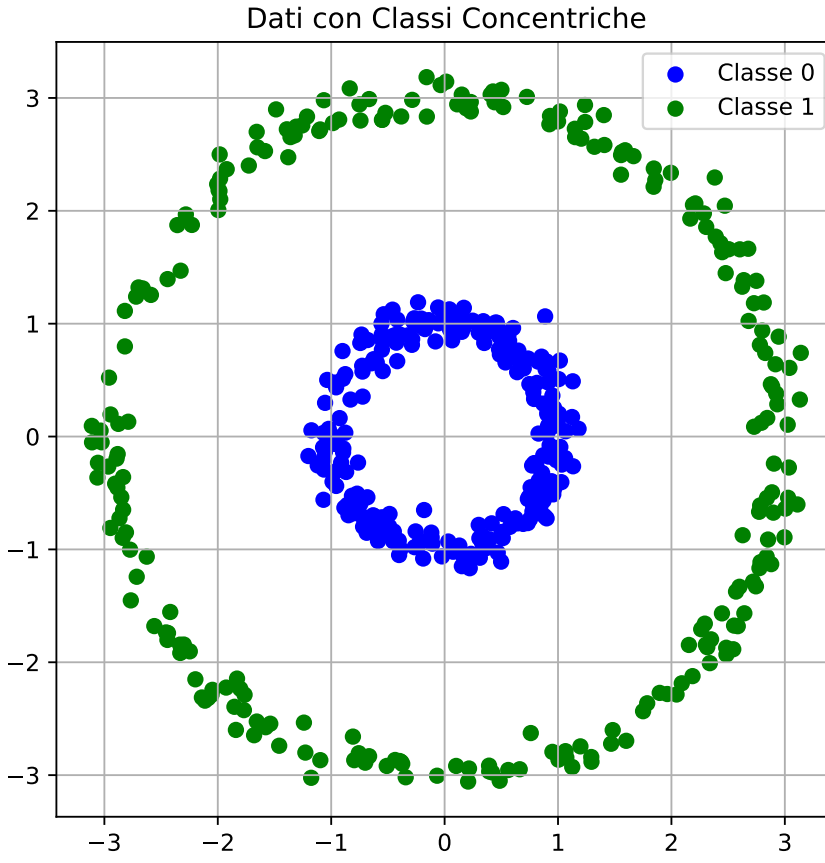


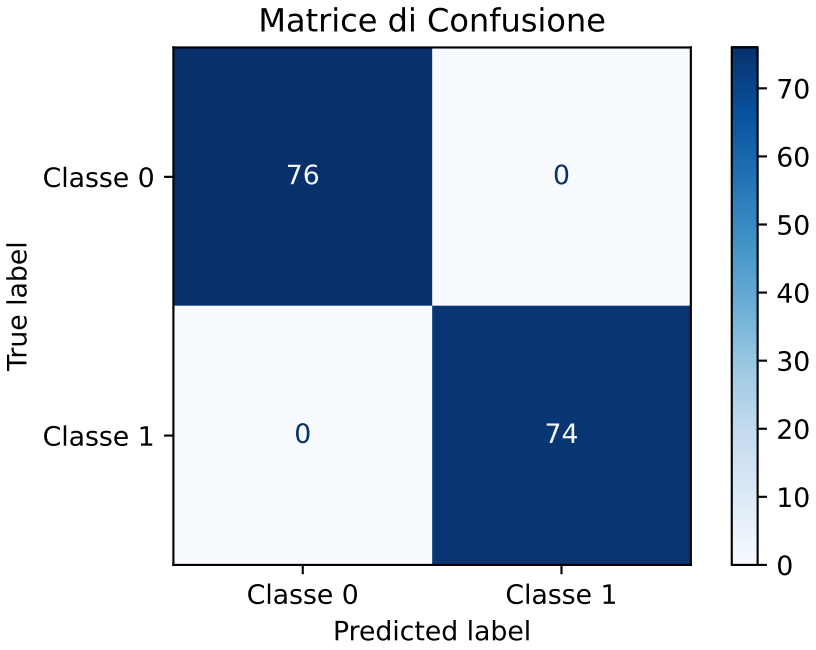

```
# Importa le librerie necessarie
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix,
 ↪  ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler

# 1. Generazione dei dati: due classi concentriche
def generate_concentric_circles(n_samples=500,
 ↪  noise=0.1):
    np.random.seed(42)
    n_samples_per_class = n_samples // 2
    angles = np.random.rand(n_samples_per_class) * 2
 ↪  * np.pi
```

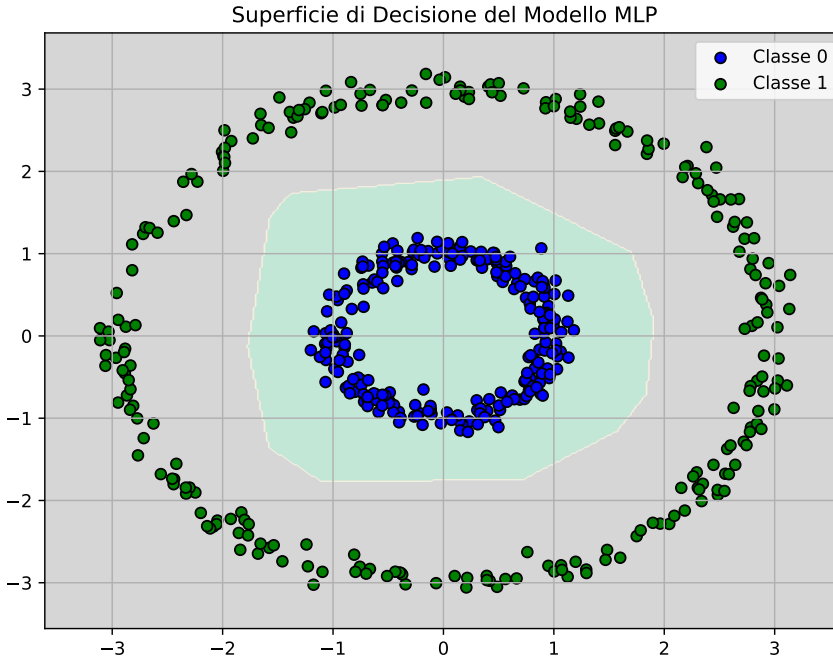
```
    inner_radius = 1 + noise *
 ↪  np.random.randn(n_samples_per_class)
    outer_radius = 3 + noise *
 ↪  np.random.randn(n_samples_per_class)

    inner_x = np.stack([inner_radius *
 ↪  np.cos(angles), inner_radius * np.sin(angles)],
 ↪  axis=1)
    outer_x = np.stack([outer_radius *
 ↪  np.cos(angles), outer_radius * np.sin(angles)],
 ↪  axis=1)

    X = np.concatenate([inner_x, outer_x], axis=0)
    y = np.array([0] * n_samples_per_class + [1] *
 ↪  n_samples_per_class)

    return X, y

X, y = generate_concentric_circles()

# 2. Visualizzazione dei dati
plt.figure(figsize=(6, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1],
 ↪  c='blue', label='Classe 0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1],
 ↪  c='green', label='Classe 1')
plt.title('Dati con Classi Concentriche')
plt.legend()
plt.grid(True)
plt.show()

# 3. Divisione del dataset e normalizzazione
X_train, X_test, y_train, y_test =
 ↪  train_test_split(X, y, test_size=0.3,
 ↪  random_state=42)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Creazione e addestramento del modello MLP
model = MLPClassifier(hidden_layer_sizes=(10, 5),
 ↪  activation='relu', max_iter=1000,
 ↪  random_state=42)
model.fit(X_train_scaled, y_train)

# 5. Valutazione: matrice di confusione
y_pred = model.predict(X_test_scaled)
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
 ↪  display_labels=["Classe 0", "Classe 1"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Matrice di Confusione")
plt.show()

# 6. Grafico della superficie di decisione
h = .02  # step size
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() +
 ↪  .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() +
 ↪  .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

grid = np.c_[xx.ravel(), yy.ravel()]
grid_scaled = scaler.transform(grid)
Z = model.predict(grid_scaled)
Z = Z.reshape(xx.shape)
```

```
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap=plt.cm.Pastel2,
 ↪  alpha=0.8)
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1],
 ↪  c='blue', label='Classe 0', edgecolors='k')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1],
 ↪  c='green', label='Classe 1', edgecolors='k')
plt.title("Superficie di Decisione del Modello MLP")
plt.legend()
plt.grid(True)
plt.show()
```


Dati con Classi Concentriche
Classe 0
Classe 1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Matrice di Confusione
Classe 0
Classe 1
True label
76
0
0
74
Classe 0
Classe 1
Predicted label
70
60
50
40
30
20
10
0



Il codice Python scritto per questo esperimento segue la seguente logica:

- **Generazione dei dati**: Due insiemi di punti sono distribuiti in cerchi concentrici: il primo (classe 0) vicino all'origine, il secondo (classe 1) su un raggio maggiore. La forma dei dati rende il problema non linearmente separabile.
- **Visualizzazione**: Il primo grafico mostra chiaramente la distribuzione circolare dei due insiemi di punti.
- **Preprocessing**: I dati vengono suddivisi in un training e test set (70% - 30%) e normalizzati con StandardScaler.
- **Modello MLP**: È stata creata una rete con due strati nascosti (10 e 5 neuroni) e funzione di attivazione ReLU. La rete viene addestrata per classificare i dati.

- **Valutazione**: Viene calcolata e mostrata la matrice di confusione, che evidenzia l'accuratezza del modello nel distinguere le due classi.
- S**uperficie di decisione**: Il terzo grafico mostra come la rete ha appreso a separare le due classi: la forma curva della regione di decisione indica che il modello ha effettivamente appreso la complessità dei dati, superando i limiti del percettrone semplice (che può solo separare linearmente).

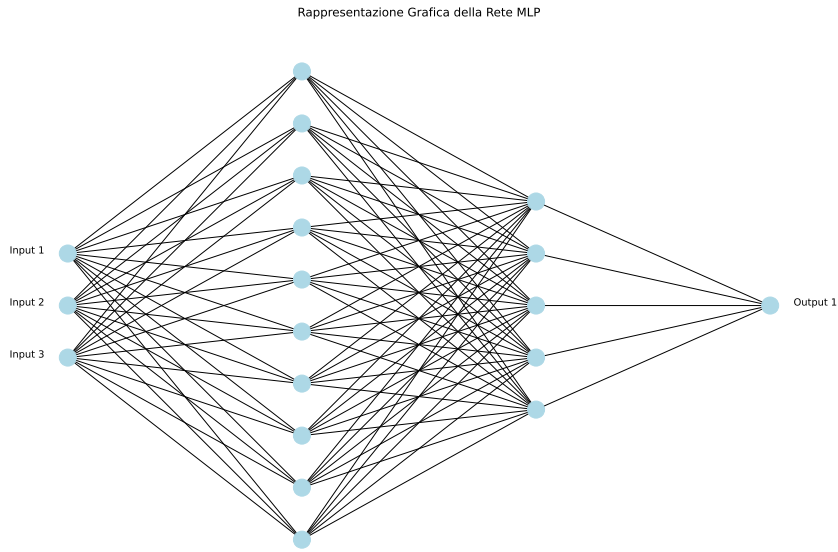
13.9.3. Esperimento 3: Rete MLP per la predizione dell'esito di un caso giudiziario

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 ingressi, 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).

Usando la funzione introdotta nell'esempio 1 possiamo disegnare la rete MLP che vogliamo adottare per risolvere il problem di classificazione in studio. Si noti che è necessario eseguire il codice dell'esempio 1 per poter eseguire il seguente codice altrimenti Python segnalerà come errore il fatto di non conoscere la funzione draw_mlp().

```
# 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)
```

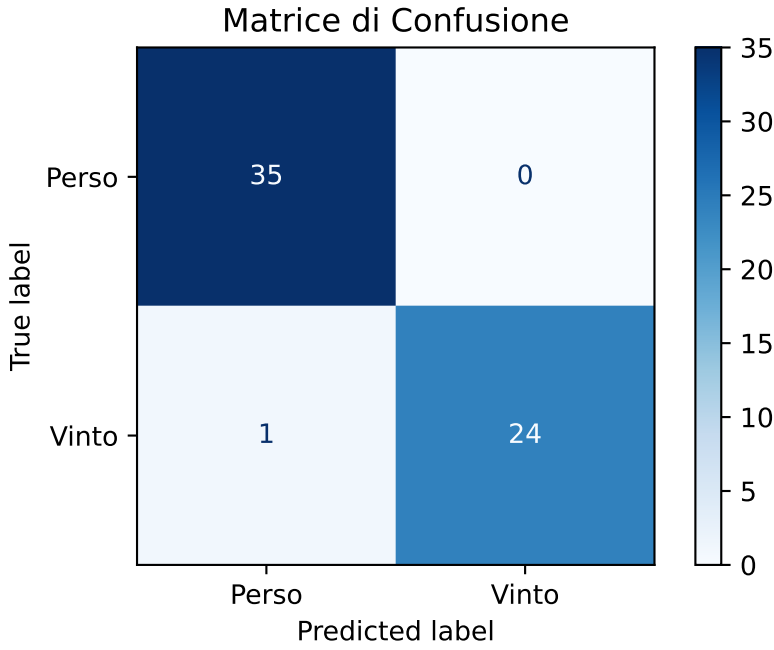


L'implementazione in Python della rete MLP per il nostro problema di classificazione è la seguente:

```
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
from sklearn.preprocessing import StandardScaler #
 ↪  Aggiunto import

# Simuliamo un dataset per predire se un caso
 ↪  giudiziario sarà vinto o perso basandosi su tre
 ↪  caratteristiche
# 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_clusters_per_class=1, random_state=42)

# 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)

# --> Aggiunta sezione per lo scaling
# Scalare i dati (buona pratica per MLP)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# <-- Fine sezione scaling

# Creare e addestrare un modello MLP usando i dati
 ↪  scalati
mlp_model = MLPClassifier(hidden_layer_sizes=(10,
 ↪  5), max_iter=1000, random_state=42)
```

```
mlp_model.fit(X_train_scaled, y_train) # Usa
 ↪  X_train_scaled

# Predire sul set di test scalato
y_pred = mlp_model.predict(X_test_scaled) # Usa
 ↪  X_test_scaled

# Mostrare la matrice di confusione usando i dati
 ↪  scalati
ConfusionMatrixDisplay.from_estimator(mlp_model,
 ↪  X_test_scaled, y_test, display_labels=["Perso",
 ↪  "Vinto"], cmap=plt.cm.Blues) # Usa X_test_scaled
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)
```



	precision	recall	f1-score	support
Perso	0.97	1.00	0.99	35
Vinto	1.00	0.96	0.98	25
accuracy			0.98	60
macro avg	0.99	0.98	0.98	60
weighted avg	0.98	0.98	0.98	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 97%, mentre per i casi vinti è dell'100%. Questo significa che quando il modello prevede un caso come "Perso", nel 97% dei casi ha ragione, mentre per i casi "Vinto", la precisione è del 100%.
    - **Recall**: La recall per i casi persi è del 100% e per i casi vinti è del 96%. Questo indica che il modello è riuscito a identificare correttamente la quasi totalità dei casi vinti e persi.
    - **F1-score**: L'F1-score, che rappresenta un bilanciamento tra precisione e recall, è del 99% per i casi persi e del 98% per i casi vinti, riflettendo una ottima performance complessiva del modello.

Osservazioni: - Il modello ha raggiunto un'accuratezza complessiva del 98%, 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.

13.10. Esercizi

13.10.1. Esercizio 1: Funzioni di attivazione non lineari

Spiega il ruolo delle funzioni di attivazione non lineari nelle reti MLP: - Perché non è sufficiente usare solo funzioni lineari?
- Cosa accadrebbe alla capacità della rete di apprendere pattern complessi?

13.10.2. Esercizio 2: Confronto tra architetture neurali

Metti a confronto CNN, RNN e GAN:

- Qual è il tipo di dati o problema che ciascuna architettura affronta meglio?
- In quali contesti applicativi (es. immagine, testo, generazione di contenuti) useresti ognuna?

13.10.3. Esercizio 3: Vantaggi dei modelli pre-addestrati

Spiega perché può essere utile utilizzare modelli pre-addestrati:

- Quali sono i principali benefici in termini di tempo, accuratezza e risorse computazionali?
- In quali situazioni l'uso di un modello pre-addestrato è particolarmente indicato?

14. Elaborazione del Linguaggio Naturale

14.1. Introduzione

Il linguaggio naturale è la lingua parlata o scritta che usiamo quotidianamente per comunicare. È ricco, complesso e carico di ambiguità. Un frammento di testo può contenere riferimenti impliciti, sinonimi, metafore e strutture grammaticali variabili. Ad esempio, la frase "Marta ha visto Luca in giardino con il binocolo" è ambigua: chi osserva con il binocolo? Chi è osservato con il binocolo? Per un sistema NLP, risolvere questa ambiguità richiede di analizzare il contesto o utilizzare modelli avanzati come i *Transformer* (Vaswani et al. 2017).

14.2. Obiettivi dell'NLP

L'elaborazione del linguaggio naturale (NLP) è un processo che acquisisce una porzione di testo, chiamata *documento* (anche se si tratta di una parola o una frase), la scompone in unità elementari dette *token* e procede con fasi di analisi, comprensione e generazione. Ad esempio, in ambito giuridico, l'NLP può estrarre clausole chiave da un contratto o generare riassunti di sentenze (Surden 2019).

14.3. Acquisizione

L'acquisizione del documento inizia con la raccolta dei dati, che possono essere testuali o audio. I dati testuali sono codificati in sequenze di caratteri, mentre i dati audio sono trasformati in campioni digitali e convertiti in testo. Al termine, il documento è pronto per l'elaborazione, memorizzabile in un file di testo o in un database.

14.4. Pre-elaborazione

La *pre-elaborazione* è fondamentale per preparare i dati testuali all'analisi, riducendo complessità, ridondanza e variabilità del linguaggio naturale. Serve a:

- *Riduzione della complessità*: testi più semplici per modelli più efficienti.
- *Migliore rappresentazione*: vettori numerici più rappresentativi del significato.
- *Migliore generalizzazione*: evita che i modelli apprendano rumore linguistico.

Le principali operazioni sono:

- *Tokenizzazione*: suddivide il testo in *token* (parole, frasi o caratteri).
  Esempio: "La giustizia è uguale per tutti." → ["La", "giustizia", "è", "uguale", "per", "tutti", "."]
- *Rimozione della punteggiatura*: elimina segni non utili all'analisi.
- *Normalizzazione del testo*: standardizza le parole, ad esempio:

  - Conversione in minuscolo.
  - Espansione di forme contratte (“don’t” → “do not”).
- *Rimozione delle stopword*: elimina parole frequenti ma poco rilevanti (“il”, “e”, “di”).
- *Stemming e lemmatizzazione*:
  - *Stemming*: riduce una parola alla radice (“giudicando” → “giudic”).
  - *Lemmatizzazione*: riduce alla forma base, considerando il contesto (“giudicando” → “giudicare”).
- *Rimozione di cifre o caratteri speciali*: utile se non rilevanti.
- *Correzione ortografica* (facoltativa): migliora la qualità dei dati.

La scelta delle tecniche dipende dall’applicazione: per analisi giuridiche o traduzioni, la *lemmatizzazione* è preferibile; per compiti generali, basta lo *stemming*. Al termine, si ottiene un documento di *token* o parole, pronti per essere convertiti in numeri.

14.5. Word Embeddings

I *word embeddings* convertono le parole in vettori numerici per l’elaborazione computazionale. Questi vettori, in uno spazio multidimensionale, catturano le relazioni tra parole in modo compatto ed efficiente, consentendo l’applicazione di algoritmi di machine learning e deep learning (Mikolov et al. 2013). Sono essenziali per attività come l’analisi del sentiment, la classificazione del testo, la traduzione, il riconoscimento di entità nominate e il *question answering*. Negli ultimi anni, numerosi articoli scientifici hanno esplorato questo tema (Semanticscholar.org).

14.5.1. Proprietà Sintattiche dei Word Embeddings

I *word embeddings* catturano informazioni sulla struttura grammaticale. Parole con ruoli sintattici simili si trovano vicine nello spazio vettoriale, riflettendo la *coerenza sintattica*. Ad esempio, la relazione tra "veloce" e "velocemente" è simile a quella tra "lento" e "lentamente", esprimibile come: `vector("velocemente") - vector("veloce") + vector("lento")  vector("lentamente")`. Questa capacità emerge dalle co-occorrenze nei corpora di testo ed è utile per il *part-of-speech tagging* e il *parsing* sintattico.

I *word embeddings* codificano anche relazioni morfologiche: "cane" e "cani" o "camminare" e "camminato" sono vicini, riflettendo una comprensione implicita della morfologia. Modelli come *Word2Vec* e *GloVe* non codificano direttamente l'ordine delle parole, ma lo catturano indirettamente tramite co-occorrenze locali. I *Transformer* (es. *BERT*) invece considerano l'intero contesto (Devlin et al. 2019).

Un'altra proprietà è la *similitudine funzionale*: parole sostituibili in una frase, come "cane" e "gatto" in "Il *cane* abbaia" e "Il *gatto* miagola", hanno vettori simili, utile per l'analisi sintattica e l'estrazione di relazioni.

Analogia	Operazione Vettoriale	Risultato Atteso
Veloce : Velocemente :: Lento :?	`vector("velocemente") - vector("veloce") + vector("lento")`	`vector("lentamente")`
Re : Regno :: Regina :	`vector("regno") - vector("re") + vector("regina")`	`vector("reginato")`

Analogia	Operazione Vettoriale	Risultato Atteso
Camminare : Camminato :: Correre :?	`vector("camminato") - vector("camminare") + vector("correre")`	`vector("corso")`

14.5.2. Proprietà Semantiche dei Word Embeddings

I *word embeddings* eccellono nel catturare il significato delle parole, posizionando termini simili (es. “cane” e “gatto”) vicini nello spazio vettoriale. La *similarità semantica* è misurabile con la similarità cosinusoidale: valori alti indicano significati affini. Ad esempio, “automobile” e “veicolo” sono più vicini di “cane” e “gatto”, riflettendo gradi di relazione.

Codificano anche relazioni *is-a* (iponimia/iperonimia): “automobile” è vicino a “veicolo”, suo iperonimo. L’analogia “re” - “uomo” + “donna” $\approx$ “regina” dimostra come catturino relazioni semantiche complesse. Gli antonimi (es. “caldo” e “freddo”) sono distanti, ma possono condividere un campo semantico (es. temperatura). Inoltre, raggruppano parole per temi: “ospedale”, “chirurgo” e “infermiere” formano un cluster sanitario.

14.5.3. Tecniche di Generazione

- *Word2Vec*: Usa reti neurali per apprendere vettori, con *Skip-Gram* (predice il contesto da una parola) e *CBOW* (predice la parola dal contesto), efficace per analogie semantiche (Mikolov et al. 2013).

- *GloVe*: Basato su una matrice di co-occorrenza globale, bilancia relazioni locali e globali (Pennington, Socher, and Manning 2014).
- *FastText*: Estende *Word2Vec* rappresentando parole come n-grammi di caratteri, ideale per parole rare e lingue morfologicamente ricche (Bojanowski et al. 2017).
- *ELMo*: Genera embedding contestuali, analizzando l'intera frase per cogliere sfumature (Peters et al. 2018).
- *Transformer* (es. *BERT*, *RoBERTa*): Usa l'attenzione per modellare dipendenze a lunga distanza, migliorando la comprensione sintattica e semantica (Devlin et al. 2019; Y. Liu et al. 2019).

14.6. Sentence Embeddings

I *sentence embeddings* rappresentano il significato di intere frasi in vettori, posizionando frasi simili vicine nello spazio vettoriale. Superano i *word embeddings*, catturando il senso di un pensiero completo, e migliorano attività come classificazione, traduzione e generazione di testo (Reimers and Gurevych 2019).

14.6.1. Proprietà dei Sentence Embeddings

I *sentence embeddings* codificano il significato complessivo della frase, superando i limiti dei singoli termini. Sono vettori densi in spazi multidimensionali, con modelli come *BERT* che catturano il contesto di ogni parola. Preservano struttura sintattica e semantica, mappando frasi di lunghezza variabile in vettori fissi per l'apprendimento automatico. Alcuni, multilingue, codificano testi di lingue diverse in uno spazio condiviso, utili per applicazioni cross-linguali.

14.6.2. Applicazioni

- *Ricerca semantica e recupero*: Cerca in base al significato, es. "migliori ristoranti" restituisce "locali di alta cucina".
- *Classificazione del testo*: Classifica articoli (es. sport, politica) in base al significato complessivo.
- *Analisi del sentiment*: Determina se una recensione è positiva o negativa, considerando l'intera frase.
- *Rilevamento di parafrasi*: Identifica frasi con lo stesso significato, es. domande duplicate in forum.
- *Question answering*: Trova risposte pertinenti confrontando similarità semantiche.
- *Traduzione automatica*: Migliora la fluidità, cogliendo il contesto della frase.
- *Clustering e modellazione di argomenti*: Raggruppa documenti simili per tema.
- *Retrieval-Augmented Generation (RAG)*: Recupera passaggi rilevanti per risposte coerenti.
- *Generazione di testo*: Produce testo contestualmente appropriato.

14.6.3. Tecniche per la Generazione di Sentence Embeddings

- *Media dei word embeddings*: Metodo base, ma perde ordine e relazioni complesse.
- *Modelli dedicati*: *Sentence-BERT*, *Universal Sentence Encoder* e *InferSent* generano embedding di alta qualità (Reimers and Gurevych 2019; Cer et al. 2018).
- *Transformer*: *BERT*, *RoBERTa* e altri usano pooling (es. media, token CLS) per sfruttare il contesto (Devlin et al. 2019).

14.7. Transformer

La comprensione e generazione del linguaggio naturale sono essenziali in ambito legale, per analizzare contratti o sentenze. Le reti neurali ricorrenti (RNN), come le *LSTM*, perdono informazioni su testi lunghi. I *Transformer*, introdotti nel 2017, rivoluzionano l'NLP con il meccanismo di *attenzione* (Vaswani et al. 2017).

14.7.1. Il Cuore dei Transformer: l'attenzione

L'**attenzione** si concentra sulle parti rilevanti di un testo. In una clausola come "Il locatore, dopo aver ricevuto il pagamento del canone mensile entro il 5 del mese, garantisce al conduttore l'uso esclusivo dei locali commerciali situati in Via Roma 10, salvo eventuali violazioni delle norme di sicurezza specificate nell'Allegato B", il modello collega "conduttore" a "locatore" e "Allegato B", dando meno peso a "mensile" o "5".

L'**attenzione** si calcola con:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- *Q (Query)*: rappresenta la parola analizzata (es. "conduttore").
- *K (Key)*: valuta la rilevanza di altre parole.
- *V (Value)*: aggrega le informazioni ponderate.
- $\sqrt{d\_k}$: normalizza i punteggi per stabilità.
- *softmax*: trasforma i punteggi in pesi.

Parola (Key)	Peso di Attenzione (ipotetico)
locatore	0.25

Parola (Key)	Peso di Attenzione (ipotetico)
locali commerciali	0.20
Allegato B	0.15
al	0.10
canone	0.05
mensile	0.02
dopo	0.01
5	0.01

Questo collega parole distanti, cruciale per contratti o pareri legali.

14.7.2. Multi-Head Attention: Vedere il Linguaggio da Diverse Prospettive

La *multi-head attention* esegue l'**attenzione** in parallelo: una testa analizza la sintassi, un'altra la semantica, per una comprensione più completa.

14.7.3. L'Architettura del Transformer: Encoder e Decoder

- **Encoder**: Trasforma il testo (es. un contratto) in vettori ricchi di contesto, con strati di *multi-head attention* e reti feed-forward.
- **Decoder**: Usa l'output dell'**encoder** e le parole generate per produrre testi, come riassunti o risposte.

14.7.4. Perché i Transformer Hanno Rivoluzionato l'NLP?

- **Parallelizzazione**: Elabora tutte le parole simultaneamente, più veloce delle RNN.
- **Dipendenze a lunga distanza**: Collega informazioni distanti, es. clausole e allegati.
- **Scalabilità**: Modelli come *BERT* e *GPT* eccellono su grandi dataset, utili per:
  - Classificazione di documenti legali.
  - Ricerca giuridica.
  - Generazione e traduzione di testi.

14.8. Large Language Models (LLM)

Un *LLM* è un modello di machine learning addestrato su vasti corpora di testo per comprendere e generare linguaggio, utile per classificazione, ricerca, generazione e traduzione (Brown et al. 2020). Modelli noti includono:

1. *GPT-4 (OpenAI)*: Denso, multimodale, contesto fino a 128K token.
2. *Grok (xAI)*: Progettato per risposte rapide e analisi di dati social.
3. *Claude (Anthropic)*: Sicuro, con contesto fino a 200K token.
4. *Gemini (Google DeepMind)*: Multimodale, contesto fino a 1M token.
5. *DeepSeek (DeepSeek AI)*: *MoE*, efficiente, 671B parametri.
6. *Qwen (Alibaba)*: *MoE*, multilingue, 128K token.

Sono accessibili come chatbot (per prompt testuali) o API (per richieste HTTP). I costi variano in base al provider, al modello, al

volume di token e alla regione. Per dettagli, consultare i siti ufficiali (es. https://x.ai/grok per Grok, https://openai.com per GPT).

Tip

Mixture of Experts (MoE) I modelli *Mixture of Experts (MoE)* dividono un problema complesso in sottoproblemi, assegnandoli a "esperti" specializzati (sottoreti neurali). Una rete di *gating* sceglie l'esperto adatto per l'input. L'*attivazione sparsa* coinvolge solo alcuni esperti, riducendo il carico computazionale pur mantenendo molti parametri (Shazeer et al. 2017).

14.8.1. Prompt Engineering

La *prompt engineering* ottimizza i prompt per ottenere risposte precise e coerenti dagli *LLM*, strutturando istruzioni, contesti ed esempi per massimizzare rilevanza e accuratezza (P. Liu et al. 2023).

14.8.1.1. Strategie di Prompt Engineering

- *Specificità del prompt*: Istruzioni chiare evitano ambiguità. *Esempio*: "Riassumi la sentenza Cass. Civ., Sez. Unite, n. 500/1999, evidenziando i punti salienti relativi all'articolo 2043 del Codice Civile in materia di danno ingiusto, e indica la ratio decidendi in non più di 200 parole."
- *Few-shot prompting*: Fornisce esempi di input/output. *Esempio*: "Esempio 1: Input: 'Clausola di riservatezza per contratto di lavoro.' Output: 'Il dipendente si impegna a mantenere riservate le informazioni aziendali, pena il

risarcimento del danno.' Ora, scrivi una clausola per un contratto di locazione che preveda un indennizzo per mancato preavviso di recesso."

- *Chain-of-thought prompting*: Esplicita i passaggi logici.
  Esempio: "Considera un caso di responsabilità contrattuale: [descrizione]. Pensa: 1. Quali sono gli elementi della responsabilità? 2. Le prove dimostrano l'inadempimento? 3. C'è un nesso di causalità? 4. Quali sono le conseguenze giuridiche? Concludi."
- *Impersonificazione di un ruolo*: Simula un professionista.
  Esempio: "Agisci come un giudice della Corte Costituzionale. Esamina la legittimità costituzionale dell'articolo X della legge Y, con un parere formale e motivato."

14.8.2. Modelli LLM "Locali"

L'uso di *LLM* via API di provider come OpenAI o Google è costoso e pone problemi di privacy. I modelli *locali* si eseguono su CPU e GPU standard, usando tecniche come:

1. *Quantizzazione*: Riduce i pesi a 8, 4 o 2 bit (*GPTQ*, *bitsandbytes*) (Dettmers et al. 2023).
2. *Distillazione*: Un modello piccolo imita uno grande (*DistilBERT*) (Sanh et al. 2019).
3. *Pruning*: Elimina pesi poco rilevanti.
4. *Low-Rank Adaptation (LoRA)*: Aggiunge matrici addestrabili ai pesi congelati (Hu et al. 2021).
5. *MoE*: Attiva solo sotto-reti rilevanti (Shazeer et al. 2017).
6. *Ottimizzazione hardware*: Usa *ONNX*, *TensorRT*, *GGUF*.

Piattaforme open source:

Piattaforma	Descrizione	Supporto modelli
llama.cpp	Esecuzione di *LLaMA* su CPU, anche mobile.	*GGUF* (*LLaMA*, *Mistral*)
MLC LLM	Compilatore per WebGPU, iOS, Android.	*LLaMA*, *Mistral*, *Qwen*
Ollama	Esecuzione semplice con un comando.	*LLaMA*, *Mistral*, *Gemma*

Piattaforme proprietarie:

Piattaforma	Descrizione	Note
OpenVINO™	Ottimizza modelli su CPU e VPU (Intel).	Supporta *ONNX*
NVIDIA TensorRT-LLM	Ottimizza *LLM* su GPU NVIDIA.	Uso enterprise

Nota: *FlashAttention* è una tecnica per ottimizzare l'attenzione, riducendo il consumo di memoria (Dao et al. 2022).

14.9. Applicazioni in Giurisprudenza

L'NLP rivoluziona il settore legale (Surden 2019; Ashley 2017):

- *Analisi legale*:
  - Revisiona contratti per clausole o ambiguità.
  - Cerca precedenti con *NER* e *topic modeling*.

  - Interpreta leggi con analisi semantica.
- *Assistenza giuridica*:
  - Chatbot per consulenza preliminare.
  - Automatizza risposte e gestione clienti.
- *Previsione di esiti*: Analizza sentenze per prevedere risultati.
- *Gestione documentale*:
  - Classifica e archivia documenti.
  - Estrae dati chiave per database.
- *Supporto alla decisione*: Aiuta i giudici con citazioni e precedenti.
- *Compliance e due diligence*: Monitora conformità e rischi.
- *Innovazione*: Crea modelli di documenti e simula negoziazioni.
- *Etica e accessibilità*: Traduce testi e semplifica il linguaggio legale.

14.10. Sfide e Considerazioni Etiche

- *Bias e neutralità*: Gli *LLM* possono amplificare pregiudizi; servono dati diversificati e de-biasing (Bender et al. 2021).
- *Privacy*: Rispettare il GDPR, anonimizzando i dati.
- *Responsabilità e trasparenza*: Chiarire chi è responsabile per errori e rendere i modelli spiegabili.
- *Accuratezza*: Verifica umana per evitare errori gravi.
- *Equità nell'accesso*: Evitare divari tra grandi e piccoli studi legali.
- *Manipolazione*: Rilevare falsi documenti generati.
- *Etica*: Bilanciare automazione e impatto sul lavoro.
- *Regolamentazione*: Collaborare per norme sull'NLP.

14.11. Conclusioni

L'NLP offre strumenti potenti per la giurisprudenza, ma richiede attenzione a etica e implicazioni legali. Questo capitolo introduce l'NLP e le sue applicazioni, preparando a ulteriori ricerche.

14.12. Laboratorio di Python

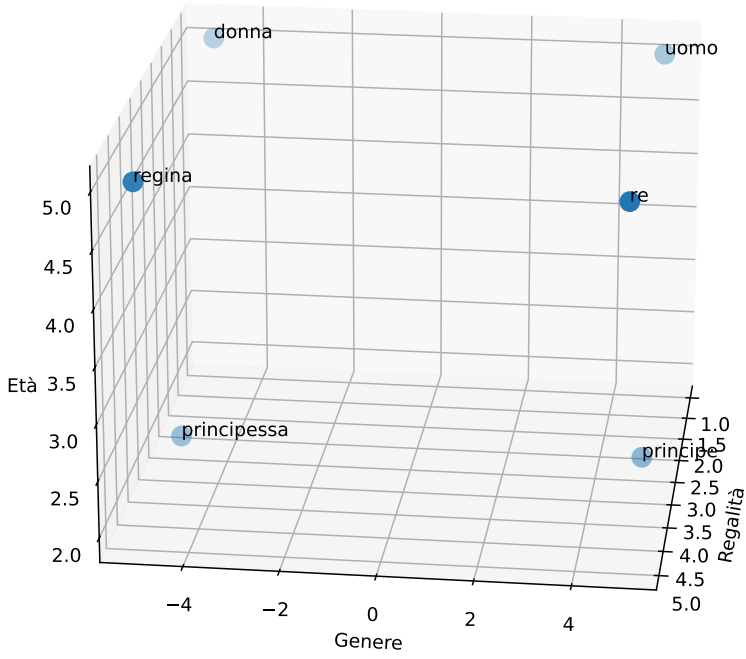
14.12.1. Esperimento 1: Introduzione ai Word Embeddings

In questo esperimento vogliamo implementare i word embedding in maniera semplificata e intuitiva per cercare di capire le notevoli proprietà che riescono a esprimere. I *word embeddings* sono vettori numerici. Qui li rappresentiamo in 3D con assi per *regalità*, *genere* ed *età* per parole come "re", "regina", ecc.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# Word embeddings semplificati
word_embeddings = {
    "re": [5.0, 5.0, 5.0],
    "regina": [5.0, -5.0, 5.0],
    "principe": [3.0, 5.0, 2.0],
    "principessa": [3.0, -5.0, 2.0],
    "uomo": [1.0, 5.0, 5.0],
    "donna": [1.0, -5.0, 5.0]
}
```

```
words = list(word_embeddings.keys())
embeddings = list(word_embeddings.values())

# Grafico 3D
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
x = [emb[0] for emb in embeddings]
y = [emb[1] for emb in embeddings]
z = [emb[2] for emb in embeddings]
ax.scatter(x, y, z, s=100)
for i, word in enumerate(words):
    ax.text(embeddings[i][0], embeddings[i][1],
 ↪  embeddings[i][2], word)
ax.set_xlabel('Regalità')
ax.set_ylabel('Genere')
ax.set_zlabel('Età')
ax.set_title('Visualizzazione 3D dei Word
 ↪  Embeddings')
ax.view_init(elev=20, azim=7)
plt.show()

# Distanza coseno
def distanza_coseno(vec1, vec2):
    prodotto_scalare = np.dot(vec1, vec2)
    norma_vec1 = np.linalg.norm(vec1)
    norma_vec2 = np.linalg.norm(vec2)
    return 1 - (prodotto_scalare / (norma_vec1 *
     ↪  norma_vec2))

# Matrice delle distanze
n_words = len(words)
matrice_distanze = [[0.0] * n_words for _ in
 ↪  range(n_words)]
```

```
for i in range(n_words):
    for j in range(n_words):
        matrice_distanze[i][j] =
 ↪  distanza_coseno(embeddings[i], embeddings[j])

# Visualizzazione
print("Matrice delle distanze coseno:")
print("          ", end="")
for word in words:
    print(f"{word:>10}", end="")
print()
for i, word1 in enumerate(words):
    print(f"{word1:10}", end="")
    for j in range(n_words):
        print(f"{matrice_distanze[i][j]:10.3f}",
         ↪  end="")
    print()

# Calcolo di "regina"
def sottrai_vec(vec1, vec2):
    return np.array(vec1) - np.array(vec2)
def somma_vec(vec1, vec2):
    return np.array(vec1) + np.array(vec2)

print("Word embedding di regina:")
print(word_embeddings["regina"])
print("Word embedding calcolato: regina = re - uomo
 ↪  + donna")
print(somma_vec(sottrai_vec(word_embeddings["re"],
 ↪  word_embeddings["uomo"]),
 ↪  word_embeddings["donna"]))
```




```
Matrice delle distanze coseno:
                re    regina  principeprincipessa
                uomo     donna
re             0.000     0.667     0.063     1.000
0.111     0.919
regina         0.667     0.000     1.000     0.063
0.919     0.111
principe       0.063     1.000    -0.000     1.316
0.137     1.273
```

```
principessa     1.000     0.063     1.316     -0.000
1.273     0.137
uomo            0.111     0.919     0.137     1.273
0.000     0.980
donna           0.919     0.111     1.273     0.137
0.980     0.000
Word embedding di regina:
[5.0, -5.0, 5.0]
Word embedding calcolato: regina = re - uomo + donna
[ 5. -5.  5.]
```

I risulrari numerici dell'esperimento ci mostrano che anche se in una versione molto semplificata, i word embedding sono in grado di esprimere proprietà come la regalità, il genere e l'età. Infatti si noti come sottraendo a **re uomo** si ottiene la regalità che può essere sommata a **donna** per ottenere **regina**.

14.12.2. Esperimento 2: Analisi del Sentiment

In questo esperimento vogliamo stimare il sentiment (molto negativo, negativo, neutrale, positivo, molto positivo) di un frammento di testo. A questo scopo usiamo un modello addestrato su più lingue per stimare il sentiment in frammenti di testo in Italiano e Giapponese.Il modello prescelto è tabularisai/multilingual-sentiment-analysis di Hugging Face.

> **Caution**
>
> Nota: Richiede pip install transformers e una connessione per scaricare il modello.

```
from transformers import pipeline
```

```
# Pipeline con modello multilingue
pipe = pipeline("text-classification",
 ↪  model="tabularisai/multilingual-sentiment-analysis")

# Testo in Italiano
frase = "Questo prodotto non è fatto bene."
risultato = pipe(frase)
print(frase)
print(risultato)

# Testo tradotto in Giapponese
frase = "         "
risultato = pipe(frase)
print(frase)
print(risultato)
```

```
WARNING:tensorflow:From
C:\Users\lcapitanio\AppData\Local\Packages\PythonSoftwareFounda
The name tf.losses.sparse_softmax_cross_entropy is
deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy
instead.

Questo prodotto non è fatto bene.
[{'label': 'Negative', 'score': 0.5313267111778259}]

[{'label': 'Negative', 'score':
0.48841869831085205}]
```

L'output dell'esperimento ci dice che entrambre le frasi sono considerate negative. Si noti come la frase in Giapponese sia la traduzione della frase in Italiano fatta da un traduttore basato su LLM e non mostrato in questo esempio.

14.12.3. Esperimento 3: Sistema di Domanda-Risposta su un Testo Giuridico

In questo esperimento vogliamo implementare un semplicissimo sistema che ci consenta di di dialogare con un breve testo giuridico. Facendo domande e ottenendo risposte sui contenuti del testo.Il modello adottato è "deepset/roberta-base-squad2" sempre da Hugging Faceper rispondere a domande su un testo giuridico.

> **Caution**
>
> Nota: Richiede pip install transformers e una connessione per scaricare il modello.

```
from transformers import pipeline

# Pipeline per question answering
qa_pipeline = pipeline("question-answering",
 ↪  model="deepset/roberta-base-squad2")

# Contesto giuridico
contesto = """
L'articolo 3 della Costituzione italiana stabilisce
 ↪  che tutti i cittadini hanno pari dignità sociale
 ↪  e sono eguali davanti alla legge,
senza distinzione di sesso, razza, lingua,
 ↪  religione, opinioni politiche, condizioni
 ↪  personali e sociali.
È compito della Repubblica rimuovere gli ostacoli di
 ↪  ordine economico e sociale che,
limitando di fatto la libertà e l'eguaglianza dei
 ↪  cittadini, impediscono il pieno sviluppo della
 ↪  persona umana.
```

```
"""

# Domande
domanda1 = "Qual è il compito della Repubblica?"
risultato1 = qa_pipeline(question=domanda1,
 ↪  context=contesto)
print(f"Domanda: {domanda1}")
print(f"Risposta: {risultato1['answer']}\n")

domanda2 = "Chi ha pari dignità sociale?"
risultato2 = qa_pipeline(question=domanda2,
 ↪  context=contesto)
print(f"Domanda: {domanda2}")
print(f"Risposta: {risultato2['answer']}\n")

domanda3 = "Quale articolo della Costituzione
 ↪  stabilisce che tutti i cittadini hanno pari
 ↪  dignità sociale?"
risultato3 = qa_pipeline(question=domanda3,
 ↪  context=contesto)
print(f"Domanda: {domanda3}")
print(f"Risposta: {risultato3['answer']}\n")
```

```
Domanda: Qual è il compito della Repubblica?
Risposta: rimuovere gli ostacoli

Domanda: Chi ha pari dignità sociale?
Risposta: tutti i cittadini

Domanda: Quale articolo della Costituzione
stabilisce che tutti i cittadini hanno pari dignità
sociale?
Risposta: L'articolo 3
```

Dall'output dell'esperimento possiamo vedere che il sistema è in grado di rispondere a domande sul testo giuridico. Ovviamente il sistema non è in grado di rispondere a domande il cui oggetto non è specificato nel testo. Inoltre, la brevità del testo usata non consente di rispondere a domande che richiedono un'analisi più approfondita.

14.13. Esercizi

14.13.1. Esercizio 1: Pre-elaborazione di un Testo Giuridico

Scegli un breve articolo di legge o una clausola contrattuale (es. dall'articolo 3 della Costituzione italiana o da un contratto di locazione). Scrivi un programma Python che applichi le seguenti operazioni di pre-elaborazione: tokenizzazione, rimozione delle stopword (usando una lista predefinita, es. da nltk), conversione in minuscolo e lemmatizzazione (con spacy o nltk). Visualizza il testo originale e quello pre-elaborato. Rifletti: quali operazioni sono più utili per preparare il testo a un'analisi semantica in ambito legale?

14.13.2. Esercizio 2: Creazione di un prompt efficace

Progetta un prompt per un Large Language Model (es. usando few-shot prompting o chain-of-thought) per generare un riassunto di una sentenza giuridica (es. Cass. Civ., Sez. Unite, n. 500/1999). Il prompt deve specificare: lunghezza massima (es. 150 parole), elementi chiave da includere (es. fatti, decisione, ratio decidendi) e tono formale. Testa il prompt con un modello accessibile (es. tramite Hugging Face o un'API gratuita) e valuta la qualità del riassunto

prodotto. Quali modifiche al prompt migliorerebbero la precisione e la chiarezza del risultato?

14.13.3. Esercizio 3: Analisi semantica con word embeddings

Modifica il codice dell'Esempio 1 del laboratorio Python per aggiungere nuove parole legate al contesto giuridico (es. "giudice", "avvocato", "tribunale", "sentenza"). Assegna loro vettori 3D basati su tre dimensioni rilevanti (es. autorità, ruolo, formalità). Calcola la similarità cosinusoidale tra queste parole e visualizza i risultati in una matrice. Estendi il codice per verificare un'analogia giuridica (es. "giudice : tribunale :: avvocato : ?"). Discuti come i word embeddings potrebbero supportare la ricerca di precedenti legali o l'estrazione di concetti da documenti giuridici.

Bibliografia

Ashley, Kevin D. 2017. "Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age." *Cambridge University Press*.

Bender, Emily M, Timnit Gebru, Angelina McMillan-Major, and Margaret Shmitchell. 2021. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610–23.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. "Enriching Word Vectors with Subword Information." *Transactions of the Association for Computational Linguistics* 5: 135–46.

Brams, Steven J., and Alan D. Taylor. 1996. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. 2020. "Language Models Are Few-Shot Learners." *Advances in Neural Information Processing Systems* 33: 1877–1901.

Buchanan, Bruce G., and Edward H. Shortliffe. 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.

Cer, Daniel, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, et al. 2018. "Universal Sentence Encoder." *arXiv Preprint arXiv:1803.11175*.

Dao, Tri, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness." *Advances in Neural Information Processing Systems* 35: 16344–59.

DeLancey, Craig. 2017. *A Concise Introduction to Logic*. Open SUNY. https://open.umn.edu/opentextbooks/textbooks/452.

Dettmers, Tim, Artem Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. "QLoRA: Efficient Finetuning of Quantized LLMs." *arXiv Preprint arXiv:2305.14314*.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–86.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 226–31.

European Parliament and Council of the European Union. 2024. "Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 Laying down Harmonised Rules on Artificial Intelligence and Amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU and (EU) 2022/2555 (Artificial Intelligence Act)." 2024/1689. *Official Journal of the European Union*. Vol. L. European Union. http://data.europa.eu/eli/reg/2024/1689/oj.

Goldman, J., and A. D. Procaccia. 2014. "Spliddit: Unleashing Fair

Division Algorithms." *ACM SIGecom Exchanges* 13 (2): 41–46.
Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. https://www.deeplearningbook.org/.
Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. "Generative Adversarial Nets." In *Advances in Neural Information Processing Systems*, 2672–80.
Governo Italiano. 2021. "Programma Strategico Per l'intelligenza Artificiale 2022-2024." Ministero dell'Università e della Ricerca; Ministero dello Sviluppo Economico; Ministro per l'innovazione tecnologica e la transizione digitale. https://assets.innovazione.gov.it/1637777289-programma-strategico-iaweb.pdf.
Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.
Haykin, Simon. 2009. *Neural Networks and Learning Machines*. 3rd ed. Pearson.
Hinton, Geoffrey E, and Ruslan R Salakhutdinov. 2006. "Reducing the Dimensionality of Data with Neural Networks." *Science* 313 (5786): 504–7.
Hotelling, Harold. 1933. "Analysis of a Complex of Statistical Variables into Principal Components." *Journal of Educational Psychology* 24 (6): 417–41.
Hu, Edward J, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. "LoRA: Low-Rank Adaptation of Large Language Models." *arXiv Preprint arXiv:2106.09685*.
Inc., Northpointe. 2018. *Correctional Offender Management Profiling for Alternative Sanctions (COMPAS)*. https://bja.ojp.gov/sites/g/files/xyckuh186/files/media/document/compas.pdf.
James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Springer.

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*.

Karpathy, Andrej. 2015. "The Unreasonable Effectiveness of Recurrent Neural Networks." https://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Kleinberg, Jon, Sendhil Mullainathan, and Manish Raghavan. 2018. "Human Decisions and Machine Predictions." *The Quarterly Journal of Economics* 133 (1): 237–93.

Kolmogorov, Andrey Nikolaevich. 1933. *Grundbegriffe Der Wahrscheinlichkeitsrechnung*. Springer.

Kuhn, Max, and Kjell Johnson. 2013. *Applied Predictive Modeling*. Springer.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. "Deep Learning." *Nature* 521 (7553): 436–44.

Lipton, Richard J., Evangelos Markakis, Elchanan Mossel, and Amin Saberi. 2004. "On Approximately Fair Allocations of Indivisible Goods." *Proceedings of the 5th ACM Conference on Electronic Commerce*, 125–31.

Liu, Pengfei, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. "Prompt Engineering for Large Language Models." *arXiv Preprint arXiv:2307.09067*.

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." *arXiv Preprint arXiv:1907.11692*.

MacQueen, James. 1967. *Some Methods for Classification and Analysis of Multivariate Observations*. University of California Press.

McCarthy, John, Marvin Minsky, Nathaniel Rochester, and Claude E. Shannon. 1955. *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*.

McDermott, John P. 1980. "RI: An Expert in the Computer Systems

Domain." In *AAAI Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:13902711.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Efficient Estimation of Word Representations in Vector Space." https://arxiv.org/abs/1301.3781.

Moulin, Herv 'e. 2003. *Fair Division and Collective Welfare*. MIT Press.

Murphy, Kevin P. 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press.

Newman, Mark EJ. 2004. "Fast Algorithm for Detecting Community Structure in Networks." *Physical Review E* 69 (6): 066133.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning. 2014. "GloVe: Global Vectors for Word Representation." *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–43.

Peters, Matthew E, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. "Deep Contextualized Word Representations." *arXiv Preprint arXiv:1802.05365*.

Procaccia, Ariel D. 2013. "Cake Cutting Algorithms." *Handbook of Computational Social Choice*, 311–30.

Reimers, Nils, and Iryna Gurevych. 2019. "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks." *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3982–92.

Robertson, Jack, and William Webb. 1998. *Cake-Cutting Algorithms: Be Fair If You Can*. A K Peters/CRC Press.

Rokach, Lior, and Oded Maimon. 2005. "Clustering Methods: A Review." *Data Mining and Knowledge Discovery Handbook*, 321–52.

Rosenblatt, Frank. 1958. *The Perceptron: A Probabilistic Model*

for Information Storage and Organization in the Brain. *Psychological Review*. Vol. 65. 6. American Psychological Association.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1986. "Learning Representations by Back-Propagating Errors." *Nature* 323: 533–36.

Russell, Bertrand. 1972. *An Inquiry into Meaning and Truth*. Routledge.

Russell, Stuart J., and Peter Norvig. 2021. *Intelligenza Artificiale. Un Approccio Moderno*. Edited by Francesco Amigoni (Ed. Italiana). Pearson.

Russell, Stuart, and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach*. Pearson.

Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter." *arXiv Preprint arXiv:1910.01108*.

Searle, John R. 1980. "Minds, Brains, and Programs." *Behavioral and Brain Sciences*.

Shazeer, Noam, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Geoffrey Hinton, Quoc Le, and Jeff Dean. 2017. "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer." *arXiv Preprint arXiv:1701.06538*.

Surde, Bhushan, and Others. 2024. "The Impact of Artificial Intelligence on Legal Practice: Enhancing Legal Research, Contract Analysis, and Predictive Justice." *International Journal of Advanced Research in Computer and Communication Engineering* 13 (1): 1082–85. https://doi.org/10.17148/IJARCCE.2024.131225.

Surden, Harry. 2019. "Artificial Intelligence and Law: An Overview." *Georgia State University Law Review* 35 (4): 1305–37.

Turing, Alan M. 1950. "Computing Machinery and Intelligence." *Mind*.

Van der Maaten, Laurens, and Geoffrey Hinton. 2008. "Visualizing Data Using t-SNE." *Journal of Machine Learning Research* 9: 2579–2605.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention Is All You Need." *Advances in Neural Information Processing Systems* 30.