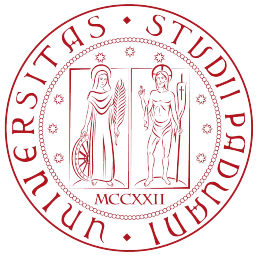

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in
Ingegneria dell'Informazione

Case-Based Reasoning: teoria ed applicazioni

Relatore:
Prof. Chiara Dalla Man

Laureando:
Samuele Viola
2013211

Anno Accademico 2022/2023

Abstract

Il Case-Based Reasoning è un'algoritmo molto usato per far prendere decisioni a un'intelligenza artificiale in situazioni più o meno complesse. La scelta si basa sul confronto tra eventi passati memorizzati all'interno di un database, che, in base a determinate caratteristiche sono riconosciuti come simili o affini. Una volta che viene stabilito quello con maggiore somiglianza, pesando le varie caratteristiche con valori numerici, l'A.I. eseguirà un azione molto simile a quella che aveva risolto il problema in quella specifica situazione. Una volta risolto, l'evento appena avvenuto e la relativa soluzione vengono memorizzati all'interno del database, per poter essere utilizzata in seguito. L'obiettivo di questa tesi è quello di approfondire questa metodologia, partendo da come è nata per poi descrivere la sua struttura, osservando esempi di applicazione reali attraverso codici Python e infine presentando la mia idea iniziale di tesi, che mi ha portato a scegliere questo argomento.

Indice

Capitolo 1

Introduzione

> «*Vivere in modo efficace è vivere con informazioni adeguate.*» (Norbert Wiener, The Human Use of Human Beings, 1950) [4]

Nella vita di tutti i giorni capita spesso di trovarsi davanti a un problema da risolvere, un imprevisto o comunque una situazione in cui bisogna fermarsi e ragionare per affrontare al meglio la difficoltà. Per farlo, l'essere umano, a volte in maniera inconscia, altre in maniera consapevole, ricorre alle esperienze passate, e alle proprie conoscenze sul settore. Si immagini, per esempio, che si sta preparando un dolce per la prima volta e non si sa bene la quantità giusta di zucchero da mettere: ne metto una quantità x_1 ma poi assaggiandola ci si accorge che manca dello zucchero. La seconda volta che si preparerà lo stesso dolce, ricordandosi che la volta precedente la quantità x_1 di zucchero era insufficiente, si metterà una quantità $x_2 > x_1$ di zucchero: chiaramente sarà più zuccherata, ma magari x_2 sarà troppo grande. Allora la terza volta ne metterò una quantità x_3 compresa tra x_1 e x_2, e così via... è chiaro che più esperienza si ha, più informazioni si memorizzano, e più il dolce che si sta preparando verrà buono. Quanto appena spiegato può essere riassunto con 3 parole: Case Based Reasoning (C.B.R.). Se si osserva il mondo, ci si accorge che questo metodo di ragionamento viene usato ovunque intorno a noi, per esempio quando scegliamo la strada più veloce in macchina, quando ordiniamo il piatto al ristorante basandoci su ciò che sappiamo ci piace, o, per prendere un esempio meno comune, quando gli avvocati, sfruttano casi precedenti per costruire argomentazioni per nuovi casi. Ci sono un'infinità di esempi. [2]

Come viene applicato nella vita di tutti i giorni, il C.B.R. viene usato moltissimo all'interno delle Intelligenze Artificiali (A.I.), perché permette al sistema non solo di trovare sempre nuove soluzioni intelligenti, ma, come farebbe un es-

sere umano, di migliorarsi sempre di più. Ha le sue radici in diverse discipline, in particolare nelle scienze cognitive, nel machine learning e più in forma generica nella filosofia. Il C.B.R. combina metodi provenienti da diversi ambiti dell'A.I., specialmente per quanto riguarda la fase di apprendimento e la risoluzione di problemi basata sull'esperienza. Ciò che distingue il Case Based Reasoning dalle altre strutture però è la capacità di sviluppare strutture specifiche per risolvere diversi tipi di problemi, come la pianificazione, la diagnosi in ambito medico, la raccomandazione di prodotti e molti altri. [3]

Nei prossimi capitoli si vedrà più nel dettaglio questo algoritmo, partendo dalle sue origini e analizzando le sue componenti e la sua struttura.

Capitolo 2

Elementi

2.1 Storia

Le origini del Case Based Reasoning si possono ricondurre all'inizio degli anni Ottanta, quando Roger Schank, professore e imprenditore americano specializzato in intelligenza artificiale [6], scrivendo lavori sulla memoria dinamica e sul ruolo fondamentale che ricordi di situazioni passate e modelli di situazione hanno nella risoluzione di problemi e nell'apprendimento, getta le basi per qualcosa che sarebbe poi diventato molto più grande. Vi sono stati poi molti altri contributi in quel periodo, come per esempio gli studi sul ragionamento analogico (ragionamento basato sull'individuazione di analogie o somiglianze tra situazioni o problemi differenti al fine di trovare una soluzione opportuna) di Gentner [14]. Più indietro nel tempo invece, Wittgenstein notò come la natura e tutto ciò che la compone è polimorfa: non esiste infatti una definizione univoca per ogni istanza, ma che anzi possono essere categorizzate in vari modi e quindi descritte da un insieme di casi [15].

A fronte di tutte queste teorie, Janet Kolodner, professore all'Università di Yale, sviluppò nel 1983 il primo sistema, ispirato al modello di memoria dinamica di Schank, in grado di effettuare dei ragionamenti con il modello C.B.R.. Chiamato CYRUS, questo sistema aveva il compito di rispondere a diverse domande sui vari viaggi e incontri dell'ex Segretario di Stato degli Stati Uniti Cyrus Vance [16].

Un altro studio che merita la menzione è quello svolto da Bruce Porter [5] e dal suo gruppo di ricerca dell'Università del Texas nel 1986, che affrontarono il problema dell'apprendimento dei concetti, dei pattern validi per compiti di classificazione, cioè con valori discreti non numerici, generalmente etichette (ad

esempio uno studente a fine anno può essere classificato come promosso o bocciato). La loro attività di ricerca portò alla creazione di PROTOS, un sistema che divenne conosciuto perché mescolava la conoscenza generale, ovvero i principi, le regole che valgono anche per gli esseri umani che si approcciano a un problema, con la conoscenza specifica, l'esperienza passata, che insegna come affrontare il problema con una soluzione adeguata. Unificando queste due forme di conoscenza permette all'intelligenza artificiale di muoversi più liberamente ed essere più flessibile e quindi risolvere i problemi in modo più efficiente. [5]

Negli anni successivi, si sono sviluppate altre "scuole" di Case Based Reasoning , applicando il metodo nelle decisioni giudiziarie, come hanno fatto Edwina Rissland e il suo gruppo di ricerca dell'Università del Massachusetts, e al contempo provando a combinarlo con altri metodi di ragionamento. Negli anni '90, l'interesse per il C.B.R. crebbe moltissimo a livello internazionale, tanto che venne fondata nel 1995 la Conferenza Internazionale sul Case Based Reasoning.

Un sistema di notevole successo realizzato col C.B.R. fu CLAVIER della Lockheed, utilizzato per la disposizione di parti compositi da cuocere in un forno industriale a convezione. Grazie alla sua capacità di ragionare su vari casi, il sistema riesce a scegliere il carico da cuocere successivo più appropriato in funzione di quelli precedenti: questo metodo ha eliminato la produzione di parti a bassa qualità e quindi gli scarti. [8] Sempre negli anni '90 il C.B.R. ha anche trovato un'ampia applicazione anche in altri settori come nelle scienze della salute e nella gestione della sicurezza strutturale.

Col passare del tempo, fino ad arrivare ad oggi, la ricerca sul Case Based Reasoning ha approfondito sempre più il tema, sviluppandolo anche in combinazione ad altri metodi, ad esempio è emerso di recente un approccio che utilizza il C.B.R. all'interno di un quadro statistico e formalizza l'inferenza [1] basata sui casi come un tipo specifico di inferenza probabilistica: questo consente di produrre previsioni dotate di un certo livello di confidenza.

2.2 Tipologie

Ci sono 3 tipi di Case Based Reasoning, distinti in funzione della rappresentazione dei casi e sul ragionamento che viene effettuato su di essi:

1. Strutturale;

[1] procedimento di deduzione delle caratteristiche di una popolazione, a partire da una rilevazione effettuata su un campione limitato di essa, per mezzo della stima dei parametri o attraverso il controllo delle ipotesi [9]

2. Testuale;

3. Conversazionale;

2.2.1 C.B.R. Strutturale

Nell'approccio strutturale, i casi vengono rappresentati secondo un vocabolario comune, quindi le esperienze passate fanno sempre riferimento allo stesso ambito. È chiaro che questo facilita il ragionamento, in quanto, una volta definito il vocabolario, il sistema si limita alla ricerca delle similitudini tra i vari casi in un "ambiente" predefinito, descritto unicamente dai vocaboli contenuti nel suo dizionario.

In questa tipologia di C.B.R., i casi possono essere rappresentati, a seconda del problema che si sta affrontando, da

- Tabelle attributo-valore: le informazioni vengono organizzate in tabelle (matrici) dove, generalmente, le colonne rappresentano gli attributi dei casi e le righe rappresentano i valori di questi attributi per ogni caso presente nel database del sistema;

- Rappresentazione orientata ad oggetti: le informazioni sono organizzate in funzione di vari oggetti definiti all'interno del caso; quindi i vari attributi sono considerati come delle proprietà degli oggetti, e i casi come istanze di queste classi di oggetti;

- Grafi: i nodi rappresentano gli attributi e gli archi descrivono le loro relazioni;

- Strutture dati complesse: come per esempio alberi, liste...

Questa tipologia è quella maggiormente usata e che verrà approfondita da questo testo.

2.2.2 C.B.R. Testuale

In questa tipologia non c'è un vocabolario comune a tutti i casi, ma questi sono rappresentati da stringhe. Il suo utilizzo è conveniente in situazioni in cui è necessario avere maggiore flessibilità e facilità di acquisizione delle informazioni e per avere un approccio più user-friendly.

2.2.3 C.B.R. Conversazionale

Per sistemi di assistenza intelligenti, call center e in tutte le situazioni in cui vi è una sequenza di domande e risposte tra un utente e un agente, viene usato il Case Based Reasoning conversazionale. Ogni caso viene rappresentato attraverso liste di domande e risposte diverse tra loro, e a seconda della risposta che viene data dall'utente, in funzione di ciò che risponde l'utente, il sistema decide quale domanda fare dopo, analizzando i casi precedenti registrati in memoria. La sua particolarità è che consente di catturare le conoscenze espresse direttamente durante le conversazioni, senza trascrivere le informazioni in una struttura predefinita come nel CBR strutturale.

Ciò che manca a questa struttura è l'elaborazione e la comprensione delle informazioni fornite dall'utente nelle conversazioni: il sistema sa come comportarsi ma non sa per quale motivo lo fa realmente. Questo richiederebbe lo sviluppo di algoritmi avanzati di analisi del linguaggio che, dopo aver identificato le informazioni più rilevanti, riesce a costruire un caso coerente da utilizzare ed è tutt'ora una sfida importante nel C.B.R. [3]

Capitolo 3

Principi di funzionamento del Case Based Reasoning

In questo capitolo verrà trattato il funzionamento del Case Based Reasoning, sia dal punto di vista logico, che dal punto di vista matematico. La prima sezione spiega il significato di similarità, grandezza fondamentale per il C.B.R. e che comparirà all'interno dell'algormitmo; la seconda sezione invece è dedicata all'algoritmo in se, in cui vengono descritte le varie fasi del Case Based Reasoning: Retrieve, Reuse, Revise, Retain.

3.1 Similarità

La similarità (dall'inglese "similarity") ha un ruolo fondamentale in moltissimi campi della matematica, e più in generale della scienza. In particolare nel Case Based Reasoning questa grandezza assume un significato importantissimo, in quanto il comportamento del sistema dipenderà proprio da questo indice. La similarità è un indice di quanto due oggetti x e y si assomigliano, in funzione di una o più determinate caratteristiche: da ciò è facile intuire una dipendenza di questa grandezza dalla distanza $d(x, y)$. Tra le tecniche basate su di essa maggiormente utilizzate per risolvere problemi di similarità, una delle più usate è la logica fuzzy. [10]

3.1.1 Logica Fuzzy

Importantissima nell'ambito dell'intelligenza artificiale, la logica fuzzy è un estensione della logica booleana in cui, data un'affermazione, si può valutare la sua

validità con valori reali anziché binari: in breve si può pesare il valore di verità di un'affermazione. Infatti a ciascuna proposizione si associa un valore compreso tra 0 e 1: maggiore è questo valore, e più vera è l'affermazione. Ad esempio, si può dire che:

- un neonato è "giovane" di valore 1
- un diciottenne è giovane" di valore 0,8
- un sessantacinquenne è "giovane" di valore 0,15

Definizione: La funzione similarità nel C.B.R., definita con la logica fuzzy, è $sim : P \times P \longrightarrow [0, 1]$, con $P \times P$ che è l'insieme che contiene tutte le possibili combinazioni di problemi che può affrontare il sistema, mentre l'insieme immagine sono ovviamente i valori di similarità.

Dato un nuovo problema p e due casi $c_1 = (p_1, s_1)$ e $c_2 = (p_2, s_2)$, con p_i che è il problema i-esimo e s_i la soluzione i-esima: il caso c_1 è più simile al nuovo problema p rispetto a c_2 se e solo se $sim(p, p_1) > sim(p, p_2)$ e si scrive $c_1 \succ_p c_2$. Per la maggior parte delle volte, il valore dell'indice di similarità è irrilevante, ma interessa solo l'ordine di preferenza dei casi. Importantissima è la scelta della funzione similarità, che deve valutare nel modo corretto quale effettivamente sia la soluzione da adottare in determinati problemi; infatti, se si avesse una $sim()$ non adatta, può succedere che l'ordine di preferenza non sia corretto e che il sistema adotti una soluzione non efficiente o addirittura inefficace. [3]

3.1.2 Proprietà della similarità

1. $0 \leq sim(x, y) \leq 1$ (normalizzazione)
2. $sim(x, x) = 1$ (l'oggetto più vicino a un x generico è sé stesso)
3. $sim(x, y) = sim(y, x)$ (proprietà commutativa)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (disuguaglianza triangolare in termini di distanza)

La distanza $d(x, y)$ introdotta a inizio capitolo è la grandezza duale alla funzione similarità, esprime sempre una relazione di vicinanza tra due oggetti, ma in questo caso, minore è la distanza e più i due oggetti sono simili. Questo giustifica la presenza della proprietà 4, perché per misure di similarità si può usare sia la distanza che la funzione sim:

- Si usa la similarità quando si sta lavorando con valori numerici e più in generale nei casi in cui il concetto di similitudine tra due situazioni è più intuitivo (ad esempio la vicinanza tra due numeri);
- si usa invece la distanza se si stanno trattando problemi di classificazione, dove quindi le caratteristiche non possono essere comparate con valori numerici in modo diretto, ma vengono confrontate utilizzando metriche di distanza.

[10]

3.2 L'algoritmo C.B.R.

Generalmente un ciclo di ragionamento basato sui casi, è suddiviso in 4 parti:

1. RETRIEVE: recuperare i casi precedenti più simili al problema che si sta affrontando
2. REUSE: riutilizzare le informazioni dei vecchi casi e le conoscenze nel campo per risolvere il problema
3. REVISE: osservare e rivedere ciò che succede utilizzando la soluzione proposta
4. RETAIN: conservare, memorizzare l'esperienza appena vissuta in modo che possa essere utile in futuro

Quanto appena descritto può essere riassunto dallo schema in figura 3.1: si nota subito che il nucleo dell'algoritmo è costituito dalle conoscenze generali e dalla memoria dei casi precedenti, senza un database che li contenga non sarebbe possibile effettuare nessuna delle 4 fasi del ciclo. In realtà la *general knowledge* ha più un ruolo di supporto, che può essere più o meno importante, a seconda della situazione e dalla tipologia di Case Based Reasoning che si sta utilizzando. Poiché i casi, come spiegato nel capitolo 2.1, contengono la conoscenza specifica, spesso la conoscenza generale ha un ruolo chiave in quanto crea un legame tra i vari casi e stabilisce un insieme di regole.

Definita la struttura principale del CBR, si può decomporre ulteriormente l'algoritmo: ogni passaggio (Retrieve, Reuse, eccetera) viene visto come un insieme di compiti che il "ragionatore" deve portare a termine. Vi è quindi una distinzione tra la metodologia orientata al processo e quella orientata al compito.

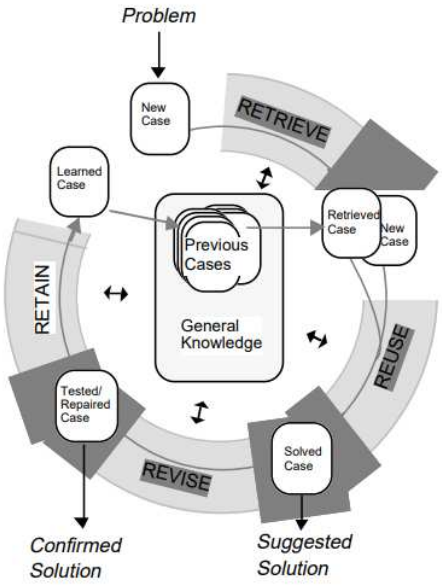

Figura 3.1: Ciclo CBR secondo Aamodt e Plaza, modificata da [5]

La visione orientata al processo, costituita dai 4 passaggi spiegati precedentemente, fornisce una panoramica esterna di come si svolge il CBR, definendone le fasi, le componenti e l'interazione tra esse. La visione orientata al compito invece, oltre che essere focalizzata sui dettagli e sulle singole operazioni da svolgere, è una visione focalizzata sul sistema o sulla persona che sta ragionando: infatti si concentra su come esso affronta ogni compito all'interno delle varie fasi. In figura 3.2 vi è un esempio di una visione orientata al compito.

Per i sistemi in generale, non c'è un metodo CBR universale per risolvere ogni problema. La sfida è trovare il metodo adatto per ogni tipologia di problema e applicazione. In accordo con quanto appena spiegato, i problemi da risolvere all'interno di un ciclo CBR sono cinque:

- Come rappresento la conoscenza?
- Che metodi uso per recuperare i casi?
- Come riutilizzo i vecchi casi?
- In che modo osservo la risoluzione del problema?
- Che metodo uso per memorizzare quanto appena osservato?

Nelle prossime sottosezioni si trovano le risposte a queste cinque domande. [5]

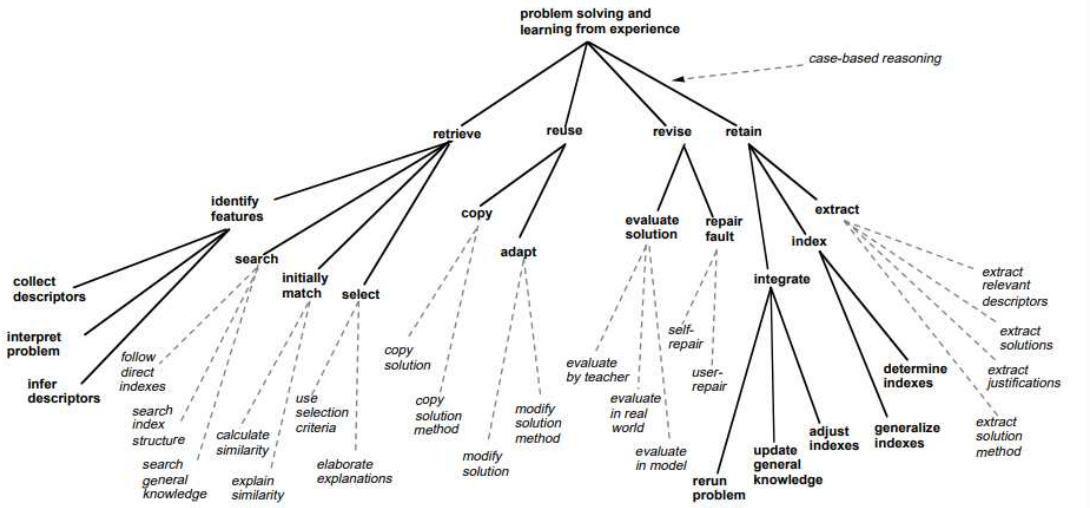

Figura 3.2: Esempio di decomposizione del CBR, tratta da [5]

3.2.1 Rappresentazione dei casi

Un sistema che ragiona secondo lo schema CBR è strettamente legato alla struttura e al contenuto della sua collezione di casi, la *case memory*. Per rendere il sistema ottimo e utilizzabile, il processo di ricerca dei vecchi casi dev'essere efficiente e rapido in termini di tempo. Inoltre, poiché, una volta risolto un problema, il sistema deve memorizzare quanto appena osservato, anche il processo di *learning* deve rispettare gli stessi requisiti. Il problema di rappresentazione di un caso consiste nel decidere quali sono le caratteristiche più importanti da memorizzare e nel trovare una struttura appropriata per descrivere i contenuti di ogni caso nella stessa maniera (CBR strutturale). Si tratta quindi prevalentemente di un problema di indicizzazione, suddiviso in due parti:

- assegnazione di indici ed etichette alle esperienze che vengono memorizzate, nei vecchi casi;

- quando analizzo delle informazioni già memorizzate, elaborare il nuovo caso in maniera dettagliata in modo da identificare gli indici.

[12]

3.2.2 Retrieve

Nella fase di recupero, vengono selezionati uno o più casi dal database, in base alla maggiore somiglianza degli indici, definita dal valore di similarità più alto (o dalla minima distanza euclidea). Vengono scelti quindi i k casi più vicini a quello nuovo basandosi quindi sulla funzione similarità definita nella struttura. [3]

In questo paragrafo verranno descritte le operazioni che deve svolgere il sistema durante la fase di Retrieve: osservando la figura 3.2 si vede che essa è divisa in quattro passaggi.

Identify Feature

Se a volte nella fase di Retrieve basta semplicemente guardare quali sono le caratteristiche affini, spesso per problemi più complessi è necessario l'utilizzo di un approccio più elaborato, specialmente per metodi in cui la *General Knowledge* è fondamentale. In PROTOS, il sistema sviluppato da J. Kolodner e di cui si è parlato nella sezione 2, se un certo input non è presente all'interno della memoria, viene chiesta all'utente un'ulteriore spiegazione che permette di collegare il nuovo dato a quelli presenti in memoria. Capire un problema significa filtrare le caratteristiche del problema rumorose, analizzare con coscenza i valori di similarità assegnati ai vari indici sono adeguati, dedurre nuove etichette (*case label*), ecc.

Search

Per cercare un insieme di casi correlati, si utilizzano le caratteristiche del problema come indici: questi possono essere utilizzati in modo diretto, utilizzandoli come puntatori o per cercare l'intera struttura indicizzata e quindi ottenendo già un insieme di casi precedenti correlati; o indiretto, analizzando le informazioni insieme alla *General Knowledge* per ricavare l'insieme desiderato. PROTOS è un mix tra ricerca diretta e indiretta: utilizza gli indici come puntatori per ipotizzare un set di candidati, e poi giustifica il match utilizzando la conoscenza generale.

Initially Match

Qui inizia il processo di ricerca di una buona corrispondenza: questo però, è diviso in due passi, il primo in cui recupera un insieme di candidati plausibili, e il processo di Select, in cui tra questa "elite" sceglie il migliore. Dopo aver ricercato tutti i casi correlati, bisogna fare delle valutazioni tra quali sono quelli migliori, è necessario definire una funzione similarità per ogni caratteristica, per

ogni indice della struttura dei casi. Un opzione per valutare la similarità è quella di attribuire un peso durante la fase di apprendimento ad ogni elemento che descrive il problema, in base alla loro importanza per la caratterizzazione del caso. Un altra opzione che permette al sistema di comprendere meglio il problema consiste nel valutare la similarità in funzione anche della conoscenza generale, arricchendo così la ricerca conoscendo obiettivi, vincoli ecc. In PROTOS ad ogni caratteristica memorizzata viene assegnato un valore di importanza per la soluzione del caso appena risolto.

Select

Una volta raggruppati i casi simili, è necessario capire la soluzione più adatta. Generalmente il caso migliore è determinato valutando più attentamente il grado di corrispondenza iniziale. Per fare ciò si cerca di giustificare la presenza di casi con caratteristiche non identiche, cercando collegamenti utilizzando la conoscenza generale. Quando una corrispondenza si rivela essere debole, il sistema prova a trovare una corrispondenza migliore che la sostituisca, seguendo collegamenti tra casi simili o correllati. Durante la fase di selezione inoltre, il sistema cerca di valutare le conseguenze di ciascun caso, cioè cerca di comprendere cosa succederebbe se applicasse la soluzione di un determinato caso al problema che deve risolvere, e fornisce, giustificandole, delle aspettative relative a quei casi. Per fare ciò spesso si ricorre a raccogliere informazioni aggiuntive dalla conoscenza di dominio generale memorizzata dal sistema oppure, come PROTOS, chiedendole all'utente. Infine i casi vengono classificati in funzione di un insieme di criteri, per esempio in base a quanto bene soddisfa i requisiti del problema, o alla sua coerenza con le aspettative, eccetera. Dopo questa classificazione, viene scelta la soluzione del caso che è più propenso, in base alla valutazione svolta, ad essere applicato al nuovo problema.

3.2.3 Reuse

Quando si utilizza la soluzione del caso scelto nella fase di Retrieve, bisogna tenere conto di due aspetti:

1. le differenze che ci sono tra vecchio e nuovo caso;
2. quale parte della soluzione scelta può essere usata per il nuovo caso.

Nelle prossime sottosezioni vengono descritti i due step, sempre facendo riferimento alla figura 3.2, che compongono la fase di Reuse.

Copy

Generalmente, nei problemi di classificazione, vengono enfatizzate le somiglianze, mentre le differenze vengono trascurate: in questo modo è sensato applicare direttamente la classe della soluzione del caso recuperato al nuovo caso senza doverla adattare. In casi più complessi però invece è necessario considerare il punto 1. di cui si è parlato nel paragrafo precedente, e quindi la parte 2. deve essere adattata, come viene spiegato nella prossima sottosezione.

Adapt

Per adattare la vecchia soluzione al nuovo caso ci sono due strade:

- *Riutilizzo trasformazionale*, che consiste nel riutilizzare la stessa soluzione del caso passato, a patto di qualche trasformazione;
- *Riutilizzo derivativo*, cioè utilizzare lo stesso metodo che è stato utilizzato per costruire la soluzione nel vecchio caso, per generare una nuova soluzione.

Nella prima tipologia, vengono utilizzati operatori di trasformazione T, per adattare la soluzione al nuovo caso in funzione delle differenze identificate nel punto 1 all'inizio della sezione. Nel riutilizzo derivativo invece, si va ad analizzare come si è arrivati alla soluzione scelta per il caso recuperato, si vanno a comprendere le giustificazioni degli operatori utilizzati, degli obiettivi considerati, dei tentativi di ricerca falliti, eccetera. Viene generato un nuovo metodo quindi ripercorrendo i passi del vecchio caso: in questa fase le alternative, gli operatori e i vari percorsi che hanno ottenuto un successo nella risoluzione vengono considerati, mentre invece quelli che hanno portato a un fallimento, per ovvi motivi, vengono evitati. Vengono perseguiti, inoltre, tutti gli obiettivi intermedi che nel vecchio caso sono stati raggiunti con successo, in modo da avere una guida per stabilire quelli per il nuovo caso: si suppone infatti che strategie simile portino a risultati simili, quindi a risolvere il caso anche nella nuova situazione.

3.2.4 Revise

In questa fase viene esaminata la soluzione ottenuta dopo aver adattato la soluzione nella fase di Reuse; è divisa in due parti:

1. valutazione della soluzione generata, se è un successo allora il sistema, nella fase di Retain, memorizzerà il caso;

2. se invece la soluzione applicata non risolve il problema, allora il sistema dovrà modificare la soluzione utilizzando le conoscenze generali.

Evaluate Solution

La soluzione scelta dal vecchio caso, copiata e adattata ora viene realmente applicata al problema. I risultati ottenuti vengono poi osservati dal sistema, per vedere se, come già anticipato all'inizio della sottosezione, il caso è stato risolto con esito positivo o negativo: questa operazione può impiegare poco o molto tempo a seconda dell'ambito, dal tipo di problema e dalla soluzione adottata. Ad esempio, se si sta valutando l'esito di un problema in ambito medico, specialmente di un sistema che fornisce supporto nelle decisioni, la fase di *Evaluate*, può durare da poche ore fino a diversi mesi. Un caso, nell'eventualità in cui dovesse volerci davvero molto tempo, può anche essere valutato in un momento intermedio, quando non è ancora stato risolto; in questo caso viene segnalato come "non valutato".

Repair Fault

Dopo aver fatto la valutazione, si è certi che la soluzione applicata abbia portato a un fallimento: come si deve comportare il sistema? In questo caso ovviamente bisogna correggere gli errori. Si incomincia rilevando quali sono stati gli elementi della soluzione che hanno portato il sistema in una "strada sbagliata", per poi andare a correggere proprio quei passi. [5]

Per capire meglio la fase di *riparazione* si può considerare CHEF, un sistema capace di realizzare ricette culinarie. Quando CHEF fa un errore, oltre a memorizzare il fallimento, cerca di generare una spiegazione a ciò analizzando le azioni svolte e le loro conseguenze, le condizioni dell'ambiente e i successivi risultati ottenuti. Questo gli permette di capire se in una determinata situazione conviene o meno eseguire una certa azione. Se individua il punto della soluzione che spiega il fallimento, per correggerlo CHEF applica il *modulo di riparazione*: dopo aver consultato le conoscenze causali generali[1] e le conoscenze di dominio[2] sistema il punto individuato aggiungendo passi aggiuntivi o modificando le sequenze di azioni svolte. [13]

[1]Il modulo di riparazione possiede una conoscenza approfondita delle relazioni causa-effetto delle azioni svolte. Questa consente al sistema di identificare le connessioni tra le azioni intraprese e gli effetti ottenuti. In questo modo, il sistema può determinare come determinate azioni potrebbero avere contribuito ai fallimenti e quali azioni dovrebbero essere adottate per evitarli

[2]Conoscenza specifica del *dominio* in cui opera, che include strategie, pratiche e tecniche comuni per affrontare situazioni simili. Può essere molto utile per suggerire modifiche coerenti all'ambito del problema

3.2.5 Retain

In quest'ultima parte del ciclo CBR si memorizza ciò che può essere utile in futuro per la risoluzione di nuovi problemi, come per esempio l'esito della fase di Revise e le eventuali correzioni della soluzione. Siccome più informazioni vengono salvate all'interno del database del sistema, e più tempo quest'ultimo impiegherà a reperire le informazioni dei vecchi casi, è importante che venga fatta un attenta selezione di cosa memorizzare.

Extract

Indipendentemente da come il problema è stato risolto, il sistema CBR aggiorna il database dei casi:

- Se è stato risolto utilizzando un caso precedente, allora ne viene costruito uno nuovo oppure il caso vecchio potrebbe essere generalizzato per includere anche quello attuale.

- Altrimenti, se il problema è stato risolto mediante altri metodi, per esempio attraverso la consultazione dell'utente, allora dovrà essere creato un caso completamente nuovo.

Cosa si utilizza come fonte di apprendimento? Che cosa è importante memorizzare? Generalmente è importante memorizzare tutte quelle informazioni che descrivono il problema (descrittori) e le soluzioni applicate in quel contesto, ma in alcuni casi può essere utile per spiegare meglio il caso, includere anche una spiegazione o una giustificazione del motivo per cui si è applicata una certa soluzione.

Infine in alcuni casi può essere utile salvare nel database anche il metodo di risoluzione del problema, che è diverso dalla soluzione perché comprende tutto il percorso di ragionamento strategico, ogni piccolo passo della soluzione, per rendere il sistema adatto per il riutilizzo derivativo.

Index

Come spiegato nella sezione 3.2.1, la rappresentazione dei casi attraverso adeguati indici è un tema centrale nel CBR. Si parla quindi di *problema di indicizzazione*, e consiste nel decidere che indici utilizzare per il recupero e come organizzare l'intera struttura indicizzata (ad esempio suddividendola in categorie eccetera).

Una soluzione banale è quella di utilizzare tutte le caratteristiche di input del problema come indici. Per quanto possa sembrare intuitivo, questo metodo potrebbe generare un sovraccarico di informazioni. Un metodo più efficace è quello di usare come indici solo le caratteristiche più rilevanti, che possono essere determinate in vari modi. Ad esempio, possono essere determinate considerando tutti i casi presenti nel database e, per ciascuno, confrontando le caratteristiche simili al problema e filtrando quelle che non sono condivise da molti casi. In questa maniera vengono ridotti rumore e ridondanza.

Integrate

Questo è l'ultimo processo della fase di Retain, nonché dell'intero ciclo di Case Based Reasoning: se nessun nuovo caso o insieme di indici è stato creato in memoria, questo costituisce il passo principale della memorizzazione. Le operazioni effettuate sono quattro.

1. **Ottimizzazione degli indici**: l'obiettivo è quello di migliorare la capacità del sistema di valutare la similarità tra i casi, per renderlo più efficiente. I vecchi indici vengono modificati in funzione del problema appena affrontato, cioè le caratteristiche che si sono dimostrate rilevanti per la risoluzione vengono rafforzate, mentre quelle che hanno portato a fallimenti o sono state pressoché irrilevanti vengono indebolite.

2. **Apprendimento delle caratteristiche rilevanti**: qui il sistema identifica le caratteristiche principali per scegliere una soluzione di successo a un determinato problema. Queste, vengono infatti collegate alle diagnosi risolutive, cioè a quei casi risolti (che sia con successo o fallimento è irrilevante) in cui queste caratteristiche hanno avuto un ruolo significativo, pesando le varie connessioni. Questi valori di intensità assegnati verranno poi regolati in base ai feedback di successo/fallimento futuri. In questo modo il sistema apprende come scegliere in modo più intelligente le caratteristiche rilevanti da salvare.

3. **Integrazione del modello di conoscenza**: per *modello di conoscenza* si intende la rappresentazione strutturata e organizzata delle informazioni, delle regole, delle relazioni e delle conoscenze relative a un certo dominio o ambito; durante la fase di *integrate* può essere utile, attraverso l'interazione tra la macchina e l'utente o con tecniche di machine learning, aggiornare questo modello con nuove informazioni.

4. **Verifica del nuovo caso**: Infine, è possibile effettuare un test del caso appena memorizzato simulando il problema iniziale.

Questi quattro punti, e più in generale la fase di integrate fanno si che il processo di memorizzazione integri in modo davvero utile la conoscenza. [5]

Dopo aver risposto alle cinque domande dell'introduzione della sezione 3.2, nel prossimo capitolo verranno presentati due sistemi che sfruttano il Case Based Reasoning.

Capitolo 4

Esempi di applicazione

In questo capitolo sono presentate due implementazioni dell'algoritmo, sia dal punto di vista pratico, scritti in Python, linguaggio molto usato nel campo dell'intelligenza artificiale.

4.1 Giocare o meno una partita a seconda delle condizioni climatiche

In questo primo esempio, viene presentato un programma, scritto da A. Marín Cantero, professore di Intelligenza artificiale presso l'Universidad Tecnológica de Bolívar, che ha lo scopo di mostrare come possono essere implementate le *4Rs* (Retrieve, Reuse, Revise, Retain). Il sistema deve risolvere un problema abbastanza semplice: stabilire se, in funzione delle condizioni climatiche esterne è possibile giocare una partita di un determinato sport all'aperto, come il calcio o il rugby.

4.1.1 Introduzione

Innanzitutto, come in ogni programma, sono state importate tutte le librerie necessarie alla scrittura. Quelle usate in questo caso sono:

- Pandas
- Numpy
- Scipy spatial distance
- Matplotlib

- Seaborn

```
import numpy as np
import pandas as pd
from scipy.spatial import distance
import matplotlib.pyplot as plt
import seaborn as sn
```

Dopo aver importato i pacchetti necessari, è il momento di capire come strutturare i casi: in questo programma sono stati implementati attraverso un file di estensione `.csv` (Comma Separated Value), cioè una lista di elementi con i valori delle varie caratteristiche separate da una virgola. La *case memory* viene descritta da `library.csv`, mentre `cases.csv` contiene un insieme di casi irrisolti cui, applicando l'algoritmo visto finora, bisogna trovare un'opportuna soluzione.

Un caso quindi è composto da quattro descrittori: *Outlook, Temperature, Humidity* e *Windy*. Determinare una soluzione per un caso consiste nel decidere il valore di *Play*, che può assumere valori booleani Yes/No, quindi consiste nel capire se il clima presente all'esterno consente di andare a giocare una partita o meno. Per caricare nel sistema queste due matrici si utilizzano le seguenti istruzioni:

```
dataset = pd.read_csv('library.csv')
futurecases = pd.read_csv('cases.csv')
dataset.dtypes
futurecases.dtypes
```

Con le prime due si sono create le due variabili contenenti i dataset, mentre con le altre due si è verificato di che tipo fossero queste, cioè di tipo *object*.

In figura 4.1b e 4.1a sono mostrati rispettivamente la case memory e la lista dei casi futuri, che sono ovviamente ancora irrisolti.

4.1.2 Codifica One Hot

Dato che una macchina non può lavorare con valori categorici, per risolvere il problema di classificazione è necessario effettuare una codifica One Hot delle caratteristiche. La codifica One Hot consiste nel rimpiazzare con valori binari, quindi 0 o 1, un valore categorico: considerato un determinato descrittore, ad esempio *Humidity*, se il valore per esempio è High, allora con questa codifica assegno 0 al valore Normal e 1 ad High. In generale se ho n valori, si associano tutti zeri tranne che per la cifra corrispondente al valore assunto dal descrittore in quel caso, a cui viene assegnato il valore 1. [17]

	Outlook	Temperature	Humidity	Windy
0	Sunny	Mild	Normal	False
1	Rainy	Cool	Normal	False
2	Overcast	Cool	High	False
3	Sunny	Cool	High	True
4	Rainy	Hot	High	True
5	Rainy	Cool	High	True

(a) Tabella casi futuri irrisolti

	Outlook	Temperature	Humidity	Windy	Play
0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rainy	Mild	High	False	Yes
4	Rainy	Cool	Normal	False	Yes
5	Rainy	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Sunny	Mild	High	False	No
8	Sunny	Cool	Normal	False	Yes
9	Rainy	Mild	Normal	False	Yes
10	Sunny	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Rainy	Mild	High	True	No

(b) Case memory

Figura 4.1

```
descriptors = dataset.iloc[:, range(dataset.shape[1] - 1)]

descriptors_oh = pd.get_dummies(descriptors)
cases_oh = pd.get_dummies(futurecases)
```

Le funzioni usate sono tutte e due del pacchetto Pandas, mentre `iloc[]` serve per costruire una nuova matrice `descriptors` partendo dal dataset iniziale, ma rimuovendo l'ultima colonna, `pd.get_dummies()` serve ad effettuare proprio la codifica One Hot vista precedentemente. [18] Un elemento di `descriptors_oh` è per esempio quello in figura 4.2. Grazie a questa codifica, ora ogni caso sarà

Outlook_Overcast	Outlook_Rainy	Outlook_Sunny	Temperature_ Cool	Temperature_ Hot	Temperature_ Mild	Humidity_ High	Humidity_ Normal	Windy_ False	Windy_ True
0	0	1	0	1	0	1	0	1	0

Figura 4.2: codifica One Hot di una riga della matrice dei descrittori

descritto da un vettore binario, che rende molto semplice l'analisi delle caratteristiche comuni: si può infatti misurare la similarità tra diversi casi usando la distanza euclidea oppure, ancora meglio, la distanza di Mahalanobis, che verrà trattata nel prossimo paragrafo.

4.1.3 Distanza di Mahalanobis come misura di similarità

Dopo aver dato una struttura comune a descrittori e casi futuri, che sia utilizzabile dal sistema, bisogna stabilire una misura di similarità tra casi. La più conveniente in questo caso è proprio la distanza di Mahalanobis in quanto basata sulle correlazioni tra i valori delle varie caratteristiche all'interno del dataset. [20]

Definizione: Dato un vettore aleatorio $\mathbf{u} = (u_1, u_2, u_3, ...u_n)$, di valore medio $\mu = (\mu_1, \mu_2, ..., \mu_n)$ la matrice di covarianza associata è

$$\mathbf{\Sigma}_u = \begin{pmatrix} var(u_1) & cov(u_1, u_2) & ... & cov(u_1, u_n) \\ cov(u_2, u_1) & var(u_2) & ... & cov(u_2, u_n) \\ ... & ... & ... & ... \\ cov(u_n, u_1) & cov(u_n, u_2) & ... & var(u_n) \end{pmatrix}$$

[21]

Definizione: La distanza di Mahalanobis tra il vettore $\mathbf{u}$ e un dataset di valore medio μ e matrice di covarianza $\mathbf{\Sigma}$ è:

$$d(\mathbf{u}) = \sqrt{(\mathbf{u} - \mu)^\top \mathbf{\Sigma}^{-1} (\mathbf{u} - \mu)}.$$

Ovviamente se $\mathbf{u}$ è di dimensione n, allora μ è anch'esso di dimensione n e $\mathbf{\Sigma}$ è nxn. [22] In questo contesto la distanza di Mahalanobis viene usata per calcolare i valori di similarità tra le caratteristiche dei casi della *case memory* e quelle dei casi futuri, quindi tra `descriptors_oh` e `cases_oh`. Più piccolo sarà questo valore, e più i due casi saranno simili fra di loro, di conseguenza verrà scelto il caso con la distanza di Mahalanobis minore.

4.1.4 Main

L'algoritmo principale è costituito prevalentemente all'interno di un ciclo for, che esegue un numero di iterazioni pari al numero di casi da risolvere (`cases_oh.shape[0]`): quindi ad ogni ciclo il sistema applicherà l'algoritmo descritto nel capitolo 3, e quindi risolverà un caso.

Innanzitutto, una volta all'interno del ciclo, vengono calcolate la matrice di covarianza del vettore dei descrittori e l'inversa della matrice di covarianza; per il calcolo di `invcovmat` è stata usata la funzione `.pinv()` del pacchetto `linalg` di Numpy [19]. Successivamente vengono creati due vettori, `actualcase` che contiene tutte le caratteristiche del caso associato alla i-esima iterazione del ciclo `for`, e `dist` che viene inizializzato con tutti zeri, ma successivamente conterrà, per ogni caso salvato nel database, la distanza di Mahalanobis tra quello e il caso attuale.

È proprio all'interno del secondo ciclo `for` che inizia la fase di Retrieve, in cui `dist` viene riempito applicando la funzione .mahalanobis() all'interno del pacchetto distance importato da Seaborn. All'interno di essa vengono richiesti 3

parametri: il caso che si sta cercando di risolvere(`actualcase`), il j-esimo caso del database (`oldcase`) e l'inversa della matrice di covarianza (`invcovmat`). Dopo aver riempito l'array, con la funzione `np.argmin()` si cerca l'indice corrispondente all'elemento di `dist` con valore più piccolo: `dataset(mindist)` è il caso scelto dopo la fase di Retrieve.

Sebbene quando era stata trattata la fase di Reuse, era stato visto che, oltre al copiare la soluzione, essa andava anche adattata (attraverso il processo di *Adapt*), in questo esempio è stata direttamente applicata la soluzione del caso più simile. Questo perché nel sistema i casi hanno esattamente la stessa struttura, e inoltre la soluzione è un valore binario, quindi non avrebbe senso adattare la soluzione al nuovo problema. La variabile `case` contiene i descrittori del caso appena risolto e nell'ultima colonna la soluzione scelta, ed è stata creata con la funzione `np.append()` che serve a concatenare più array. Successivamente viene mostrato un messaggio con la soluzione del caso i-esimo.

In questo semplice esempio la fase di Revise viene un po' trascurata: consiste nell'analizzare le *covariance heat map* correlate ad ogni iterazione del ciclo for principale. Una heat map è una mappa che rappresenta delle matrici attraverso i colori, più sono scuri e più è basso il valore dell'elemento a_{ij} e viceversa: è possibile vederne un esempio in figura 4.3. Applicando l'istruzione `sn.heatmap` di Seaborn alla matrice covarianza di `descriptors_oh` vengono quindi generate queste mappe e successivamente salvate nella cartella `/output`.

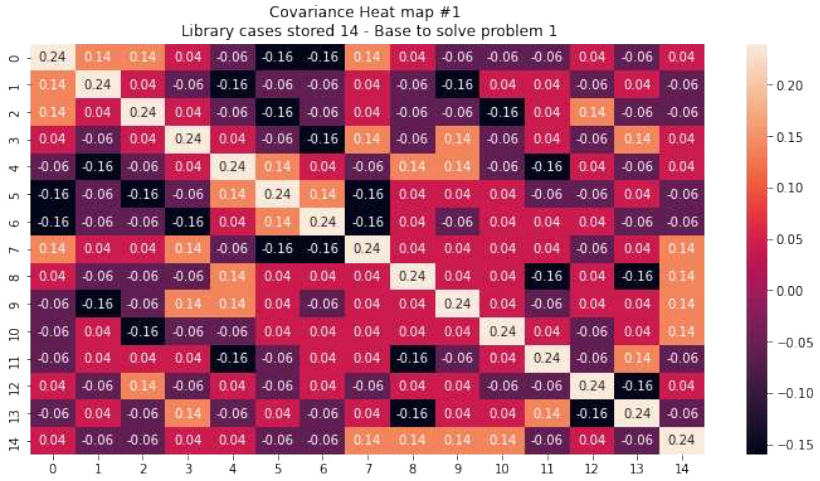

Figura 4.3

Infine la soluzione trovata viene salvata all'interno del dataset e della variabile contenente i descrittori `descriptors_oh` (Retain). L'algoritmo descritto fin'ora viene poi ripetuto fino a quando sono finiti i casi da risolvere. Dopodiché la

variabile `dataset` viene convertita in un nuovo file `.csv`, recata sempre all'interno della cartella `/output`.

```
for i in range(cases_oh.shape[0]):
    covmat = descriptors_oh.cov()
    invcovmat = np.linalg.pinv(covmat)

    actualcase = cases_oh.loc[i, :]
    dist = np.zeros(descriptors_oh.shape[0])


    for j in range(descriptors_oh.shape[0]):

        oldcase = descriptors_oh.loc[j, :]
        dist[j] = distance.mahalanobis(actualcase, oldcase, invcovmat)

    mindist = np.argmin(dist)

    case = np.append(futurecases.iloc[i, :], dataset.iloc[mindist, -1])

    print(f'> For case/problem {i}: {futurecases.iloc[i, :].to_numpy()}, solution is {case[-1]}')

    case = pd.Series(case, index = dataset.columns)
    dataset = dataset.append(case, ignore_index = True)

    sn.heatmap(np.cov(descriptors_oh, bias = True), annot = True, fmt = 'g')
    plt.gcf().set_size_inches(12, 6)
    plt.title(f'Covariance Heat map #{i} \n Library cases stored {j} - Base to solve problem {i}')
    plt.savefig(f'output/covariance_heat_map_{i}.png', bbox_inches='tight')
    plt.close()


    descriptors = dataset.iloc[:, range(dataset.shape[1] - 1)]
    descriptors_oh = pd.get_dummies(descriptors)

dataset.to_csv('output/dataset.csv', index = False)
```

4.2 Agente immobiliare

In questo capitolo viene presentato un ulteriore esempio leggermente più complesso rispetto a quello di prima, ma sicuramente più realistico e interessante. Il codice usato è una rivisitazione di un programma scritto da un utente sul web[1], e ha lo scopo di determinare il prezzo di una casa che deve essere messa in vendita.

4.2.1 Introduzione

Quando un agenzia immobiliare deve mettere in vendita la casa di un nuovo cliente, deve ovviamente fare una valutazione completa dell'abitazione per poter decidere a quale prezzo venderla. Per farlo, gli agenti immobiliari generalmente confrontano le proprietà simili che sono state vendute più recentemente e cerca di dare alla casa da vendere un prezzo simile. Se questo metodo vi sembra familiare non siete nel torto, si tratta proprio del Case Based Reasoning.

Per determinare se due case sono simili, ci si basa su una serie di criteri, come per esempio l'indirizzo, la superficie abitabile, la valutazione energetica eccetera. Queste caratteristiche hanno pesi diversi, perché alcune sono più importanti di altre, come per esempio il prezzo di vendita, che è la più utile (nonché quella da determinare). A causa dell'inflazione immobiliare, dopo ogni tre mesi, al prezzo di base viene aggiunto un ulteriore 3%.

Questo programma è stato testato su un database di 10 case a Bristol, UK ed è risultato avere una precisione del 99.2%. [23]

Per lo sviluppo del codice sono state importate semplicemente 2 liberie, Pandas, molto comoda per leggere i database e *locale*, che in questo caso serve prevalentemente a formattare il prezzo delle case utilizzando la valuta locale.

```
1 import pandas as pd
2 import locale delle case con la valuta locale
3 locale.setlocale(locale.LC_ALL, '')
```

4.2.2 La classe house

In questo codice, per definire le caratteristiche delle case, che sono salvate rispettivamente in:

- `Database.csv` per i casi già risolti memorizzati all'interno del sistema,

[1] Aaron Cardwell, per ulteriori informazioni confrontare la bibliografia al riferimento numero [23]

- `House.csv` per la casa il cui prezzo è incognito,

è stata creata la classe *house*. Una classe, in programmazione, è un costrutto che permette di definire un modello con determinate caratteristiche, metodi associati, eccetera, che tutti gli oggetti appartenenti a quella classe condivideranno. [24] In Python, per definire una classe si usa il comando `class` seguito dal nome della classe, poi successivamente le varie caratteristiche vengono elencate all'interno della sezione introdotta da `def __init__(`*parametri vari separati da virgola*`)`, specificando per ciascuna di essa il nome associato, il tipo di file e in che parametro trovare trovare il valore di quel descrittore.

```
class house:

    def __init__(self, row):
        self.name = row[0]
        self.price = float(row[1])
        self.date = int(row[2])
        self.distance = float(row[3])
        self.area = int(row[4])
        self.rooms = int(row[5])
        self.bedrooms = int(row[6])
        self.detached = row[7]
        self.garage = row[8]
        self.energy = str(row[9]).strip('\n')

        # aggiusta il prezzo a causa dell'inflazione (+3% ogni
    mese)
        for i in range(0, self.date, 3):
            self.price += self.price * 0.03
        # Initial value setting
        self.value = 0
```

In questo esempio, come si può notare, le caratteristiche associate a un oggetto della classe house sono: nome, prezzo, data in cui è stata messa in vendita la casa (utile per calcolare l'inflazione), distanza, area, numero di stanze, numero di camere da letto, la classe di energia e due variabili booleane che indicano se è presente il garage e se l'oggetto è una casa singola. Infine il ciclo for, come scritto nel commento, serve a far aumentare il prezzo a causa dell'inflazione.

4.2.3 Funzioni

Per rendere più ordinato e comprensibile il main, sono state create delle funzioni, nelle due sottosezioni verranno analizzate.

relativeValue(house)

Questa funzione, in breve, permette di calcolare il valore relativo di una casa di cui non si conosce il valore (`customerHouse`) in funzione di un'altra casa di cui invece si conosce il prezzo, in quanto è già presente nel mercato immobiliare (`house`). Questo valore verrà poi usato sia nella fase di Retrieve, per scegliere quale casa presente in `casememory` sarà la più simile, che in Reuse, per applicare e adattare la soluzione scelta dal database al nuovo caso. È importante però fare una distinzione concettuale tra similarità e questo `relativeValue`, infatti, mentre la funzione similarità indica quanto simili sono tra loro due casi, il valore restituito da questa funzione tiene anche conto dell'aspetto economico[2], in modo tale da poter appunto essere utilizzato anche per calcolare il valore effettivo della casa. Di conseguenza, al contrario della similarità, il valore economico relativo può anche essere maggiore di 1.

```
def relativeValue(house):

    adj = 0
    energyRating = {
    "A": 6,
    "B": 5,
    "C": 4,
    "D": 3,
    "E": 2,
    "F": 1
    }
    weights = {
    "distance": 4,
    "area": 2,
    "rooms": 2,
    "bedrooms": 2,
    "detached": 3,
    "garage": 1,
    "energy": 1
    }
    ideal = sum(weights.values())
    if house.distance > 0.25:
        print(f"\nHouse {house.name} troppo distante, ignorata")
        house.relativeValue, house.price = 0, 0
    else:
```

[2]se la casa dal valore incognito è molto simile a una casa presente nel database, ma ha qualche valore aggiunto, il valore restituito sarà maggiore di 1

```
26          print(f"\nHouse {house.name} entro le distanze,
    calcolando il valore relativo...")
27          rv = weights["distance"]
28          if house.area and customerHouse.area:
29              rv += weights["area"] * (house.area / customerHouse.
    area)
30          if house.rooms and customerHouse.rooms:
31              rv += weights["rooms"] * (house.rooms / customerHouse
    .rooms)
32          if house.bedrooms and customerHouse.bedrooms:
33              rv += weights["bedrooms"] * (house.bedrooms /
    customerHouse.bedrooms)
34          if house.energy and customerHouse.energy:
35              rv += weights["energy"] * (energyRating[house.energy]
     / energyRating[customerHouse.energy])
36          if house.detached == 'Y':
37              rv += weights["detached"]
38          if house.garage == 'Y':
39              rv += weights["garage"]
40          if customerHouse.detached == 'N':
41              adj += weights["detached"]
42          if customerHouse.garage == 'N':
43              adj += weights["garage"]
44
45
46          house.relativevalue = round(value / (maxvalue - adj), 2)
47          print(f"Il valore relativo della casa   : {house.
    relativeValue}")
```

Per calcolare il *relative value*, viene assegnato ad ogni attributo un coefficiente (all'interno di `weigths`) che servirà a indicare quanto importante è all'interno della valutazione quella caratteristica: vengono quindi pesate le caratteristiche. Dopodiché, attraverso una sequenza di `if`, una per ogni attributo, vengono calcolati i rapporti tra i valori dell'oggetto `house` e tra quelli di `customerHouse`, moltiplicati per il valore opportuno di `weights`, e man mano che viene fatta quest'operazione per ogni caratteristiche, `rv` viene incrementato del valore appena calcolato.

La variabile `adj` serve a ridurre il valore relativo della casa se ci sono alcune caratteristiche indesiderate, ad esempio nel caso in cui il garage dovesse essere assente o la casa non fosse separata da altre abitazioni. Questo dettaglio si può osservare alla fine della funzione, dove `adj` viene utilizzata appunto con lo scopo di aggiustare il campo `relativeValue`, attraverso la riga:

```
house.relativeValue = round(value/(maxvalue - adj), 2).
```

saveHouse(file, house)

Questa funzione serve a memorizzare il caso appena risolto: una volta assegnato un prezzo alla nuova casa, applicando questo metodo il valore scelto verrà memorizzato insieme a tutti i descrittori del caso all'interno del file `Database.csv`.

```
1  def save(file, house):
2
3      house.name = len(datahouse) + 1
4      house.price = round(house.price)
5      house.energy = house.energy + "\n"
6
7      house = list(house.__dict__.values())
8      house.pop()
9
10     outputString = ','.join(str(x) for x in house)
11
12     with open('Database.csv', 'a') as databaseOut:
13
14         for line in databaseIn:
15             line = ','.join(str(x) for x in line)
16             if outputString.split(',', 1)[1] == line.split(',',
       1)[1]:
17                 print("Exact house already in database, not
       saving...")
18                 break
19
20         else:
21             print("House not already in database, saving...")
22             databaseOut.write(outputString)
```

4.2.4 Main

In questa sezione viene presentato la parte principale del programma, il *main*, dove verranno utilizzate le due funzioni appena presentate per calcolare il valore economico di una casa da mettere in vendita, applicando il Case Based Reasoning. Ricordando i 4 step dell'algoritmo viene analizzato per primo il Retrieve.

Retrieve

Dopo aver caricato i database dei casi vecchi e di quello nuovo rispettivamente nelle variabili `casememory` e `houses`, viene creata una lista, chiamata `datahouse`, inizializzato come vuoto, e successivamente riempito, attraverso un ciclo for, con tutte le case presenti nella database sottoforma di oggetti *house*.

Dopodiché, per ogni casa presente, utilizzando la funzione `relativeValue(house)` descritta precedentemente, si calcola il coefficiente di prezzo relativo confrontando le caratteristiche di ciascuna casa del database `casememory` (numero di stanze, superficie eccetera) con quella oggetto dell'analisi, e salva i valori in un'altra lista `valueTotals`. Infine, una volta riempita questo vettore, bisogna cercare la casa più affine a quella di cui bisogna calcolare il prezzo: per farlo basta cercare nell'array `valueTotals` il valore più vicino a 1. L'indice, relativo alla variabile `datahouse` della casa scelta è salvato in `bestMatchIndex`.

```
casememory = pd.read_csv('Database.csv')
houses = pd.read_csv('House.csv')
customerHouse = house(houses.iloc[0])
customerHouse.price = 0
casememory = casememory.values.tolist()
# Definisce il vettore con le case da confrontare:    un array di
    elementi di tipo house
datahouse = []
for row in casememory[1:]:

    datahouse.append(house(row))


# Il ciclo for serve a calcolare il valore di prezzo relativo di
   ogni casa
valueTotals = []
for house in datahouse:
    relativeValue(house)
    valueTotals.append(house.relativeValue)


bestMatchIndex = valueTotals.index(min(valueTotals, key=lambda x:
   abs(x-1)))
```

Reuse e Revise

Nella fase di Reuse si applica la soluzione del caso scelto, adattandola in maniera adeguata. In questo caso per farlo è bastato prendere il prezzo della casa scelta, e dividerlo per il suo valore relativo, come mostrato nell'istruzione seguente:

```
customerHouse.price = datahouse[bestMatchIndex].price / min(
    valueTotals, key=lambda x:abs(x-1))
```

Infine viene mostrata la soluzione scelta.

```
print(f"""
--------------------------------------------------------------------------------

Closest match: House {datahouse[bestMatchIndex].name}
Relative weighted value: {datahouse[bestMatchIndex].value}
--------------------------------------------------------------------------------

Estimated customer house value: {locale.currency(customerHouse.
    price, grouping=True)}
--------------------------------------------------------------------------------

""")
```

Retain

L'ultima parte del ciclo è quella in cui il sistema salva le informazione, come già visto precendentemente. La particolarità di questo codice è che consente all'utente di scegliere se salvare il caso nel database o meno: viene posta la domanda e l'utilizzatore seleziona la risposta schiacciando Y per il sì e N per il no. Se si sceglie di salvare la casa, viene applicata la funzione saveHouse presentata nel paragrafo precedente.

```
userSave = ""
while userSave.lower() != "y" or userSave.lower() != "n":
    userSave = input("Save valuation to database? (Y/N): \n")
    if userSave.lower() == "y":
        saveHouse('Database.csv', customerHouse)
        break
    elif userSave.lower() == "n":
        print("Not saving")
        break
    else:
        print("Invalid input, enter Y for yes, or N for no")
```

Capitolo 5

Conclusioni

Il Case Based Reasoning, specialmente negli ultimi anni, è stato un argomento molto studiato e discusso in molte conferenze e riviste scientifiche del settore. Grazie a ciò, si è raggiunto un certo livello di conoscenza dei principi e tecniche di utilizzo, che mettono d'accordo la maggior parte dei ricercatori. Questo trova riscontro non solo nel numero di articoli pubblicati sull'argomento, ma anche nella pubblicazione di numerosi manuali come per esempio il libro di David B. Leake [2]. Inoltre sono stati realizzati moltissimi sistemi di intelligenza artificiale e strumenti commerciali che vengono utilizzati e che sono disponibili sul mercato, come per esempio DXplain, sviluppato dalla Harvard Medical School, che fornisce supporto nelle diagnosi mediche utilizzando il CBR. Oltre a questa, sono notevoli i campi di applicazione attuali del Case Based Reasoning: il machine learning, l'e-commerce, l'ingegneria del software e molti altri. [25]

I *trend* di ricerca scientifica principali riguardanti il CBR possono essere distinte in quattro principali tipologie.

1. ELABORAZIONE A ELEVATO PARALLELISMO: per sistemi a elevato parallelismo si intendono quei sistemi con un'architettura composta da centinaia o migliaia di blocchi CPU che lavorano contemporaneamente in parallelo. Questo permette alla macchina di elaborare un'enorme quantità di dati, anche in continua espansione. [26]

   Applicare il Case Based Reasoning a una struttura di questo tipo permette di velocizzare notevolmente la fase di Retrieve, con l'obiettivo di ottenere un sistema intelligente che, nonostante la notevole dimensione del database in costante crescita, è comunque rapido a recuperare le informazioni. Questa direzione potrebbe anche trarre beneficio dall'integrazione di reti neurali nel sistema, come stanno facendo alcuni progetti giapponesi.

2. INTEGRAZIONE CON ALTRI METODI DI MACHINE LEARNING: un campo di ricerca molto popolare è quello che riguarda l'integrazione del CBR con altri metodi di apprendimento. La costruzione di sistemi multi-strategia è forse il più ampio campo di ricerca dell'apprendimento automatico. Combinare tra di loro diversi metodi di ragionamento in maniera uniforme, consente di ottenere un sistema molto più potente ed efficiente, grazie anche alla suddivisione dei compiti per le varie strutture di apprendimento.

3. UNIONE CON ALTRI COMPONENTI DI RAGIONAMENTO: per integrazione di diversi metodi di ragionamento si intende l'utilizzo di più stili o modi di utilizzare le informazioni contenute nel database del sistema, questo fa in modo che vi sia una conoscenza generale più approfondita e flessibile all'interno della macchina, come avviene per esempio in CASEY, un sistema sviluppato da Koton nel 1989. Questo trend ha l'obiettivo di scoprire sempre nuove tecniche di acquisizione della conoscenza generale all'interno di sistemi CBR.

4. RICERCA SUGLI ASPETTI COGNITIVI: studiare e focalizzarsi più sugli aspetti cognitivi che sugli aspetti di tipo informatico può portare a una rivoluzione dei metodi di Machine Learning. Basta dare uno sguardo al passato per capire che la maggior parte delle teorie utilizzate tutt'ora nel campo dell'intelligenza artificiale (incluso ovviamente il Case Based Reasoning), devono la loro esistenza a neuroscienziati che hanno gettato, molti anni prima della nascita dei primi sistemi intelligenti, le basi per una scienza tutta nuova da scoprire. [5]

Il CBR ha portato una ventata d'aria fresca e ottimismo nel campo delle intelligenze artificiali. Con un funzionamento che è davvero simile a quello della mente umana, ha la capacità di rendere alcuni sistemi, come per esempio quelli che forniscono assistenza tecnica, davvero efficienti. L'ampia ricerca che viene effettuata al giorno d'oggi sull'argomento, ha il potenziale di portare sempre più avanzamenti nel campo delle intelligenze artificiale. Guardando al futuro, si possono immaginare svariati utilizzi alternativi dell'algoritmo, tra cui appunto l'assistenza informatica intelligenze e le *emozioni artificiali*, ovviamente sarà solo il tempo a dettare quale sarà il futuro di questo algoritmo.

Bibliografia

[1] Janet L. Kolodner., *An Introduction to Case-Based Reasoning*, 1993.

[2] "*Case-Based Reasoning: Experiences, Lessons, & Future Directions*: Edited by David B. Leake. AAAI Press/MIT Press, Menlo Park, CA/Cambridge, MA. (1996). 420 Pages. " Computers & Mathematics with Applications (1987), vol. 33, no. 4, 1997, pp. 128–128

[3] *Case-Based Reasoning - Introduction and Recent Developments*, Ralph Bergmann, Klaus-Dieter Althoff, 2009

[4] https://le-citazioni.it/autori/norbert-wiener/

[5] A. Aamodt, E. Plaza (1994); *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches*. AI Communications. IOS Press, Vol. 7: 1, pp. 39-59.

[6] https://en.wikipedia.org/wiki/Roger_Schank

[7] https://en.wikipedia.org/wiki/Case-based_reasoning

[8] *Applying Case-Based Reasoning to Manufacturing*, David Hinkle and Christopher Toomey, AI Magazine Volume 16 Number 1 (1995) (© AAAI)

[9] https://www.treccani.it/vocabolario/inferenza/

[10] Finnie, Gavin, and Zhaohao Sun. "Similarity and Metrics in Case-Based Reasoning." International Journal of Intelligent Systems, vol. 17, no. 3, 2002, pp. 273–87, https://doi.org/10.1002/int.10021.

[11] https://it.wikipedia.org/wiki/Logica_fuzzy

[12] Janet L. Kolodner, *Case-Based Learning*, Springer New York, NY 1993

[13] Kristian J. Hammond, *CHEF: A Model of Case-based Planning*, Department of Computer Science, Yale University, 1986

[14] D. Gentner: Structure mapping - a theoretical framework for analogy. *Cognitive Science*, 1983.

[15] Wittgenstein, L. (1953) *Philosophical investigations*. Blackwell

[16] Janet Kolodner: Maintaining organization in a dynamic long-term memory. *Cognitive Science*, Vol.7, s.243-280. 1983.

[17] Yefferson A. Marín Cantero, *Case-based Reasoning System*, Artificial intelligence course, Systems Engineering Program, Universidad Tecnológica de Bolívar, Cartagena de Indias, D.T. y C - Bolívar, 1p-2020, https://github.com/yammadev/cbrs/blob/master/cbrs.ipynb

[18] https://pandas.pydata.org/docs/user_guide/index.htmluser-guide

[19] https://numpy.org/doc/stable/user/index.htmluser

[20] https://it.wikipedia.org/wiki/Distanza_di_Mahalanobis

[21] L. Finesso, *Lezioni di Probabilità*, Edizioni Libreria Progetto Padova, quinta edizione, 2022

[22] Paolo medici, *Definizione 8: La distanza di Mahalanobis*, 30/05/2023; https://www.ce.unipr.it/ medici/geometry/node40.html: :text=La%20distanza%20di%20Mahalanobis%20permette,di%20scala%20automatico%

[23] https://github.com/AaronEC/CBR-House-Price-Recommender/tree/main

[24] Cay Horstmann, *Concetti di informatica e fondamenti di Java*, settima edizione 2020, Maggioli Editore

[25] Klaus-Dieter Althof, *Case Based Reasoning*

[26] Andrew Stuart Tanenbaum, *MPP: processori massicciamente paralleli*, in *Architettura dei calcolatori. Un approccio strutturale*, Milano, Pearson Education, 2006