UNIVERSITY OF TRIESTE

DEPARTMENT OF ENGINEERING AND ARCHITECTURE

Master Thesis in
Mechanical Engineering

DEVELOPMENT OF A HYBRID DECISION SYSTEM BASED ON DOMAIN KNOWLEDGE AND MACHINE LEARNING TECHNIQUES

Supervisors:

Prof. Dario Pozzetto

Eng. Helmut Kirchner

Student:

Marco Ferrari

Academic Year 2018/2019

Intelligence is the ability to adapt to change.

STEPHEN HAWKING

Prefazione

Oggigiorno l'*Intelligenza Artificiale* (AI) e la ricerca in tale ambito stanno diventando sempre più popolari. La presenza dell'AI nella vita quotidiana è ormai consolidata (i video consigliati di piattaforme come *Netflix* e *YouTube*, gli assistenti virtuali degli *smartphone* o i prodotti suggeriti nei negozi online come *Amazon* ne sono un chiaro esempio) e i compiti che possono essere efficacemente svolti aumentano sia nel numero che nel grado di complessità. Basti pensare che recentemente un software di *machine learning* sviluppato da *DeepMind* ha battuto il campione mondiale di Go [1], un gioco strategico da tavolo molto diffuso nell'Asia orientale nel quale sono possibili 2.08×10^{107} posizioni.

È senza dubbio stimolante sapere che esistono applicazioni di questa tecnologia emergente - ma già consolidata - anche nell'ambito dell'ingegneria meccanica [2], aprendo numerose strade all'innovazione. In letteratura si trovano studi che coprono molte aree: ad esempio, l'uso dell'apprendimento automatico per l'analisi vibrazionale e la ricerca di guasti in macchine rotative [3] o per l'ottimizzazione delle tecnologie di produzione additiva [4]. Ancora, l'AI può essere utilizzata nelle simulazioni fluidodinamiche [5] e nella ricerca dei design ottimi [6].

Diverse tematiche di intelligenza artificiale sono state trattate anche durante i miei studi: dalle logiche fuzzy per i sistemi di controllo alle reti neurali per la costruzione di superfici di risposta, passando per gli algoritmi genetici utilizzati come strumento per l'ottimizzazione. Sono state proprio queste digressioni dal concetto classico di ingegneria meccanica che hanno catturato la mia attenzione e mi hanno fatto appassionare a tale disciplina.

Concludendo, la scelta di redigere una tesi di laurea magistrale in ingegneria meccanica concentrandosi sullo sviluppo di un sistema di intelligenza artificiale deve essere dunque vista come un'opportunità che mi ha permesso di coniugare quanto appreso durante gli anni di università con quello che - probabilmente - sarà il futuro della ricerca e del mondo industriale.

D'altronde, «*esiste un modo migliore per fare le cose - bisogna solamente trovarlo*.» (Thomas Edison).

Indice

Abstract

This thesis is the result of my traineeship at Cybertec S.r.l., a software house based in Trieste, Italy. Its main product is *CyberPlan*, an APS - *Advanced Planning & Scheuling* - solution for the management and optimization of the supply chain. This work is the first step of a much broader research project, which final aim is to create an AI virtual assistant integrated in CyberPlan, which will help users in the identification of the most interesting elements that need some attention.

The problem in the supply chain domain is the lack of representative data: it is not possible to build a commonly shared dataset for all manifacturing companies - because of the different context, production rules, etc. -, and hence no machine learning algorithm can be use out of the box.

The solution to this problem is to overcome the scarsity of data with the domain knowledge: this means that the decision system would perform its reasoning based on some rules, derived from domain experts. This way, it is possible to make predictions also when no labelled data are available. Two main advantages are clear:

- it is possible to know how and why a certain prediction has been made. Most machine learning algorithm are, on contrary, black-boxes;
- there is no need for training data.

A such system, called in literature knowledge-based system, is basically an expert system. The drawback of this kind of systems is that they lack in adaptivity. So, in this work, we have developed an adaptive KBS, by taking advantage of some machine learning techniques. In particular, the system is able to ask the user for some feedback, and then adapt the knowledge base in order to increase the accuracy of its predictions.

This work is divided in four chapters. In the first, an introduction to the world of AI is given, its terminology is presented, and the state-of-the-art of hybrid decision systems are presented. In the second chapter, the architecture of the novel system here developed is discussed. In the third chapter, I describe the toy problems used for the validation of the system; then, a supply chain use-case is presented and studied. In the final chapter, a summary of the advantages and weaknesses of the systems are discussed.

Introduzione

Questo progetto di tesi è frutto del lavoro svolto negli ultimi sei mesi presso Cybertec S.r.l., una *software house* con sede a Trieste che propone ai suoi clienti *CyberPlan*, una soluzione APS (*Advanced Planning & Scheduling*) per la gestione e l'ottimizzazione della *supply chain*. Il tirocinio è stato possibile grazie al progetto *Talent Acquisition* promosso dall'Università degli Studi di Trieste, che ha permesso l'incontro tra le aziende del territorio e gli studenti meritevoli del Dipartimento di Ingegneria e Architettura.

Dall'incontro tra la curiosità, l'interesse e la passione per l'intelligenza artificiale dello scrivente e la visione a lungo termine di Cybertec nell'adottare soluzioni smart per il loro prodotto, è nata questa tesi. Questo ambizioso progetto è stato realizzato in stretta collaborazione con un altro tesista - nonché collega - Domagoj Korais, laureando in *Data Science and Scientific Computing* all'Università degli Studi di Trieste.

Lo scopo ultimo del progetto di ricerca è quello di realizzare un assistente alla produzione che, durante la sessione di lavoro, sappia individuare e consigliare agli utenti di *CyberPlan* gli elementi che meritano attenzione. Il grosso problema che si presenta per tale progetto è la scarsità di dati: essendo ogni contesto lavorativo a sé stante, è impossibile usare efficacemente gli algoritmi di apprendimento automatico puro per risolvere il problema.

La soluzione è quella di compensare la scarsa disponibilità di dati attraverso la conoscenza: questo significa istruire il sistema con delle regole prestabilite basate sull'esperienza di esperti di dominio, in modo che sia in grado di prendere delle decisioni senza mai aver visto dei dati. Il grande vantaggio di questo approccio (che negli ultimi anni sta diventando anche un *hot topic* di ricerca) è il fatto che, basandosi su regole prestabilite, il sistema è in grado di:

- spiegare il perché della decisione (a differenza della maggior parte degli algoritmi di *machine learning*, che forniscono i risultati a scatola chiusa - le cosiddette *black-boxes*);
- effettuare delle previsioni senza aver bisogno di dati per l'addestramento.

Il passo ulteriore che è stato fatto è quello di rendere adattivo il sistema: a seconda del feedback che riceve dall'utente, calibra le sue decisioni in modo da fornire risultati sempre più precisi e tarati sulle esigenze dell'utente. In questo passo risiede l'ibridizzazione del sistema, dove si coniugano i sistemi basati sulla conoscenza con tecniche di apprendimento automatico, per sfruttare i punti di forza di ciascuno dei due mondi.

Questo lavoro di tesi è diviso in quattro capitoli. Nel primo viene inquadrato il mondo dell'intelligenza artificiale, introducendo la terminologia e le nozioni utilizzate nel prosieguo della tesi. Alla fine del capitolo viene presentato lo stato dell'arte dei sistemi decisionali ibridi, che coniugano la conoscenza e l'apprendimento automatico. Nel secondo capitolo si passa a descrivere, sulla base delle definizioni del capitolo precedente, l'architettura del sistema sviluppato e gli algoritmi utilizzati per rendere il sistema adattivo. Nel terzo capitolo vengono presentati i problemi utilizzati per la validazione del sistema, al fine di valutarne le prestazioni, e i risultati ottenuti. Viene costruito anche un problema *ad hoc* relativo alla *supply chain*, in modo da dimostrare l'applicabilità del sistema al contesto per il quale è stato sviluppato. Infine nel quarto capitolo vengono riassunti i punti di forza del sistema e sono analizzate le possibili aree di miglioramento. A conclusione del lavoro si presentano quelli che si ritiene siano i possibili sviluppi e le future applicazioni.

Contesto e visione

Di seguito vengono delineati alcuni aspetti sociali, demografici ed economici al fine di inquadrare meglio il contesto nel quale questo progetto si colloca.

Gli studi demografici e i dati associati prevedono in Europa e soprattutto in Italia uno scenario di regressione demografica per il prossimo secolo [7].

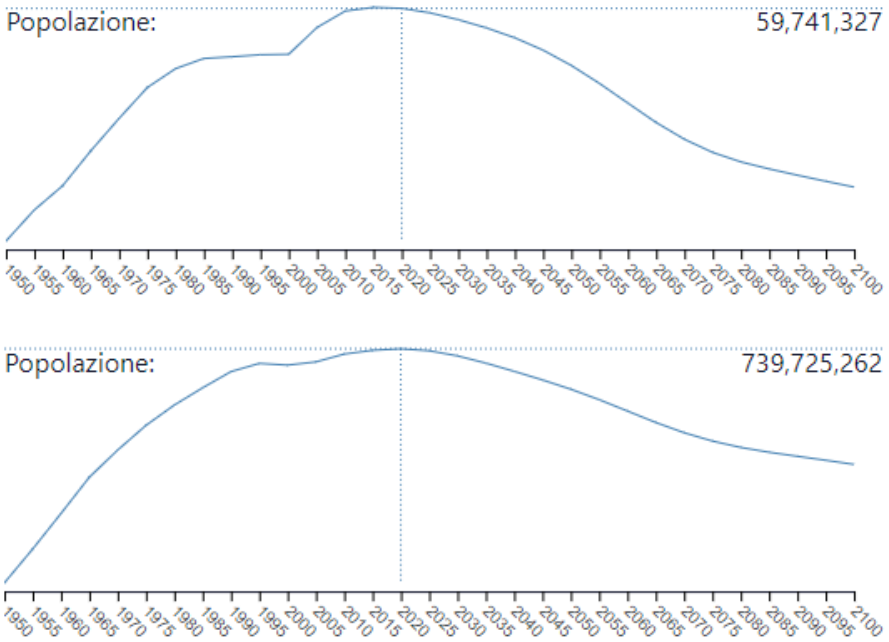


FIGURA 1: Previsione della popolazione in Italia (in alto) ed in Europa (in basso) [8].

Unitamente a ciò, nel mondo occidentale si assiste ad un progressivo declino dell'intelligenza media, mentre questa aumenta nei Paesi in via di sviluppo [9]. Tale scenario ha come ripercussione una riduzione delle persone capaci di svolgere adeguatamente i lavori che richiedono abilità cognitive sempre più ampie.

Lo studio di [10] afferma che lo sviluppo di soluzioni di intelligenza artificiale (AI) e la loro diffusione sul territorio comunitario potrebbe accrescere di circa 2700 miliardi di euro (+19%) il PIL europeo entro il 2030, con ricadute positive anche sull'occupazione.

Appare evidente quindi che anche nella realtà industriale l'intelligenza artificiale deve trovare una sempre più vasta applicazione, al fine di soddisfare i bisogni e le aspettative dei consumatori, ma anche per rilanciare l'economia e creare nuove opportunità lavorative.

Capitolo 1

Introduzione all'Intelligenza Artificiale

Questa tesi affonda le sue radici nel campo dell'Intelligenza Artificiale. Nel primo capitolo viene dunque presentato tale ambito di ricerca, focalizzando l'attenzione su concetti e strumenti utilizzati per affrontare il lavoro. In particolare, gli strumenti esaminati hanno in comune il fatto di poter essere utilizzati in applicazioni di classificazione e/o di *decision making*, che come si vedrà risulteranno essere gli argomenti principali del progetto.

I primi paragrafi del capitolo riportano le definizioni dei concetti utili ai fini della tesi e ne tratteggiano le caratteristiche che permetteranno, nel Capitolo 2, di definire i requisiti del sistema che dovrà essere sviluppato per poter risolvere il problema in esame, discusso nel Capitolo 3.

1.1 Definizioni e classificazione

L'Intelligenza Artificiale (AI - *Artificial Intelligence*) è un vasto campo di ricerca, molto attivo nell'ultimo decennio, la cui nascita risale alla metà del secolo scorso (1956).

Esistono svariate definizioni di intelligenza artificiale, molte delle quali sono complementari tra loro ed analizzano la materia da diversi punti di vista. Nella Figura 1.1 sono riportate alcune definizioni che si soffermano sia sulle *capacità cognitive* (ragionamento), che sul *comportamento*; tali aspetti inoltre sono coniugati secondo approcci basati sul comportamento *umano* e sul *raziocinio*[1].

[1]Con tale distinzione non si intende che gli esseri umani siano irrazionali nel senso di emozionalmente instabili, bensì che non siano perfetti [11].

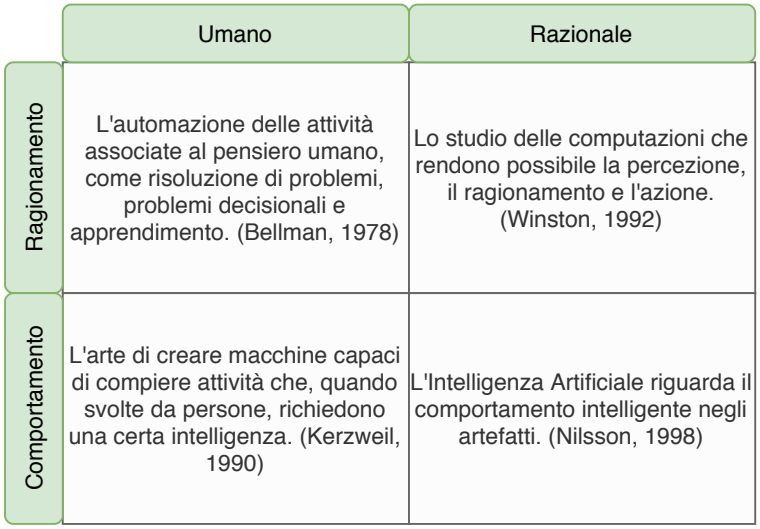


FIGURA 1.1: Alcune definizioni di Intelligenza Artificiale, organizzate in quattro categorie [11].

Tra le definizioni si può annoverare anche quella di Alan Turing, che nel 1950 sviluppò il famoso *test di Turing*, con lo scopo di dare una soddisfacente definizione operativa di intelligenza[2].

La gran parte di queste definizioni però - e il nome stesso di AI - fa riferimento al concetto di «intelligenza», senza mai definirla propriamente; lo spettro del comportamento intelligente (Figura 1.2) permette di distinguere i diversi processi che sono riconosciuti come intelligenti. I comportamenti di basso livello includono le reazioni istintive, mentre quelle di alto livello richiedono conoscenze specifiche in domini altrettanto specifici. Tale rappresentazione dei comportamenti intelligenti è utile per tracciare i progressi dell'AI.

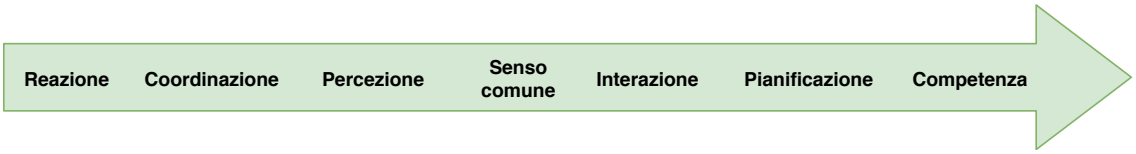


FIGURA 1.2: Spettro del comportamento intelligente. Il verso della freccia indica comportamenti intelligenti di livello crescente.

Come evidenziato da [13], i casi estremali dello spettro sono stati affrontati con successo sia attraverso tecniche convenzionali di calcolo (comportamenti di basso livello), che grazie alle prime soluzioni di AI (alto livello); è stato dimostrato

[2]Il test descrive un esperimento che permette di stabilire se una macchina è in grado di esibire un comportamento intelligente equivalente o *indistinguibile* da quello di un essere umano [12].

invece che i comportamenti intermedi di Figura 1.2 sono i più difficili da emulare in un computer.

La ricerca nel campo dell'AI ha contribuito nel corso degli anni allo sviluppo di strumenti computazionali che hanno trovato applicazione in molti settori. Limitandosi ai campi ingegneristici e a quanto è utile per il prosieguo di questo lavoro, si distinguono tre macro-categorie [13]:

- **knowledge-based systems**: sono sistemi che basano il loro funzionamento sulla *conoscenza* di un dominio specifico. A loro volta si possono distinguere:
  - *rule-based systems*, nei quali la conoscenza viene elicitata e rappresentata in forma di semplici regole;
  - *example-based systems*, nei quali la conoscenza viene ricavata da algoritmi che analizzano i dati passati come esempi al sistema;
- **computational intelligence**: ne fanno parte le logiche fuzzy, reti neurali, algoritmi genetici e algoritmi di ottimizzazione;
- **hybrid systems**: sono una combinazione dei sistemi appartenenti alle due categorie precedenti.

Si procede ora con la definizione e l'analisi più approfondita di alcuni di questi sistemi.

1.2 Sistemi basati sulla conoscenza

I sistemi basati sulla conoscenza, detti *Knowledge-Based Systems* (KBS) si differenziano da un programma convenzionale per la diversa struttura: in un KBS infatti la conoscenza di dominio è nettamente separata dal motore di applicazione della stessa, al posto di essere cablata all'interno delle righe di codice del software[3]. Tale separazione esplicita della conoscenza dalla parte relativa all'applicazione consente di rendere più agevole l'aggiunta di nuove regole o la loro modifica, sia durante la fase di sviluppo che durante la fase di uso.

[3]Si pensi ad un semplice programma convenzionale e ai costrutti *if-else* presenti nel codice. In questo caso non vi è distinzione tra motore e conoscenza.

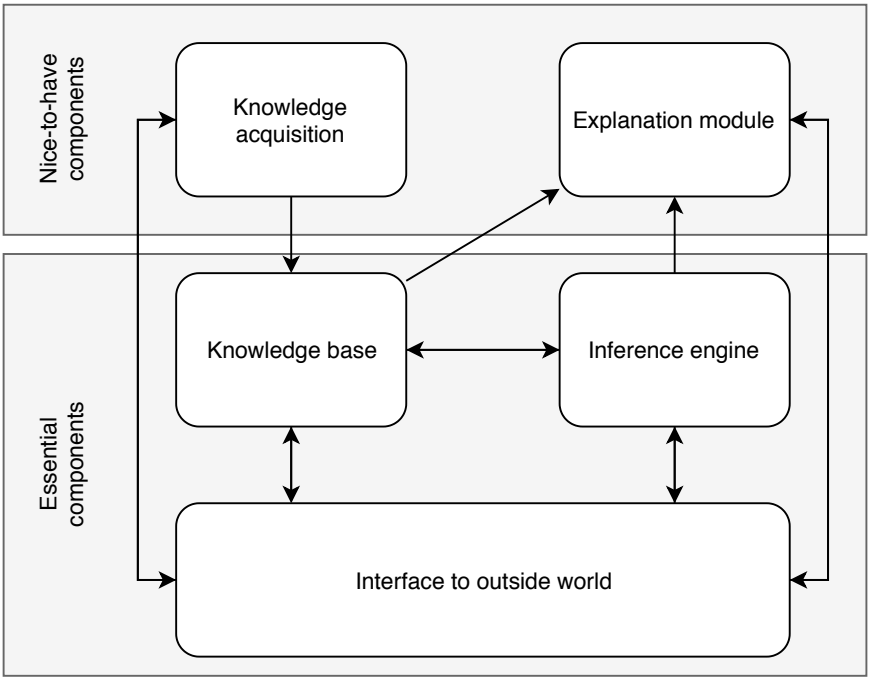


FIGURA 1.3: Architettura di un KBS e componenti principali [13].

Come schematizzato in Figura 1.3, i componenti principali di un sistema basato sulla conoscenza sono [13, 14]:

- **knowledge base**: consente la rappresentazione della conoscenza e il suo salvataggio in memoria. Il metodo più utilizzato per tale scopo è quello di utilizzare delle *regole di produzione* (anche dette regole *if-then*). Il vantaggio di regole di questo tipo è che possono spesso essere scritte in una forma molto simile al linguaggio naturale: ciò conferisce una buona interpretabilità alla *knowledge base*;

- **motore di inferenza**: è il componente responsabile dell'applicazione delle regole, quindi del ragionamento. Si distinguono motori di:

  – *deduzione*: si trova l'effetto derivante da determinati fatti forniti in input. Chiamato anche *forward-chaining*:

  $$fatti + regola \rightarrow conseguenza$$

  – *abduzione*: dato l'effetto, si trovano i fatti che possono averlo generato. Chiamato anche *backward-chaining*:

  $$conseguenza + regola \rightarrow fatti$$

  - *induzione*: si deduce una nuova regola date sia le cause che gli effetti:

$$fatti + conseguenza \rightarrow regola$$

- **interfaccia**: è il modulo che permette al sistema di interfacciarsi con il mondo esterno, cioè con umani, strumenti hardware, dati o altri software;
- modulo di **acquisizione della conoscenza**: consente l'acquisizione di conoscenza. Utile quando si costruisce una nuova *knowledge base*;
- modulo di **spiegazione**: è il componente che permette di fornire una spiegazione sul *come* e *perché* il sistema sia giunto ad una certa conclusione.

Regole e fatti

I sistemi basati sulla conoscenza coinvolgono due entità - *regole* e *fatti* - che sono utilizzate nella costruzione della *knowledge base* e nell'applicazione della stessa, rispettivamente. Si definiscono dunque [13]:

- **regole**: sono costituite da *antecedenti* (la parte «if») e da *conseguenti* (la parte «then»);
- **fatti**:
  - *noti*: sono l'input del sistema. Una regola viene attivata quando i fatti noti corrispondono alle condizioni antecedenti della regola;
  - *derivati*: sono l'output derivante dall'applicazione delle regole sui fatti noti. Si noti che un fatto derivato potrebbe rappresentare un fatto noto per l'applicazione di un'altra regola.

Nel mondo dei sistemi basati sulla conoscenza, un'assunzione comunemente accettata è la cosiddetta *ipotesi del mondo chiuso*, secondo la quale qualsiasi condizione non nota (ovvero qualsiasi condizione non contenuta nei fatti) è assunta essere falsa.

Tabelle di decisione

Una tabella decisionale è una mappa che rappresenta la relazione tra combinazioni di condizioni e combinazioni di azioni [15]. Quindi, una tabella decisionale è una rappresentazione delle regole e, dunque, della conoscenza. Un semplice esempio viene riportato in Tabella 1.1.

Condizioni						
C1	Piove?	si	si	no	$*$	no
C2	Fa freddo?	si	no	si	$*$	no
C3	Sta migliorando?	no	no	no	si	no
Azioni						
A1	Non uscire	●				
A2	Prendi il cappotto		●	●		
A3	Aspetta				●	
A4	Esci					●

TABELLA 1.1: Esempio di una tabella decisionale. Il simbolo $*$ indica una condizione *don't care* [15].

Come si vede, l'uso di tabelle decisionali permette di esprimere in maniera chiara, esplicita e compatta un set di regole che altrimenti potrebbe essere più complicato da comprendere. Da notare l'uso del carattere speciale $*$, che indica una condizione *don't care*: non importa cioè che valore assume la condizione - essa sarà sempre verificata.

Si introducono ora alcune definizioni [15]:

- due regole sono dette *contraddittorie* quando esse condividono le stesse condizioni, ma hanno azioni differenti;
- due regole sono dette *ridondanti* quando condividono le stesse condizioni ed azioni;
- una tabella si dice *meccanicamente perfetta* quando non contiene regole contraddittorie e ridondanti, e ha un numero di regole pari alla combinazione dei valori che le condizioni possono assumere.

Le tabelle decisionali - chiamate anche tabelle logiche - hanno trovato grande applicazione nel campo di sviluppo software, specialmente nella parte di documentazione; grazie all'immediatezza di comunicazione, si sono rivelate superiori ai diagrammi di flusso, pseudo-codice e altre tecniche.

Si noti che le tabelle decisionali meccanicamente perfette possono facilmente diventare molto grandi: si pensi ad una tabella con 10 condizioni, ciascuna delle quali con 5 possibili valori. Le regole possibili sono date dalle combinazioni dei 5 valori possibili per ciascuna delle 10 condizioni, cioè in questo caso da $5^{10} = 9.765.625$ regole, numero piuttosto elevato che inficia la leggibilità della tabella stessa. Per risolvere tale problema, è stata introdotta la condizione di *don't*

care. Da porre attenzione al fatto che l'uso di condizioni *don't care* ha il vantaggio di rendere più compatta la tabella, ma lo svantaggio di introdurre eventuali ridondanze e conflitti nel set di regole che devono essere opportunamente gestiti [15].

1.2.1 Sistemi esperti

I sistemi esperti, detti *Expert Systems* (ES), sono un tipo di KBS sviluppato per incorporare l'esperienza e la conoscenza di un essere umano esperto relativa ad un dominio specifico. Come si vede nella Figura 1.4, la ricerca sui sistemi esperti e sui KBS è piuttosto attiva anche se la loro nascita risale a qualche decennio fa.

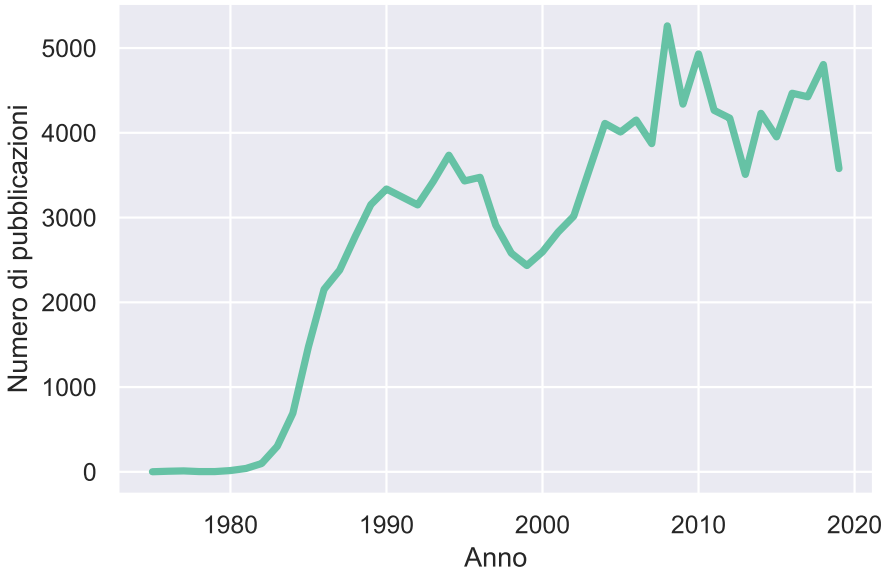


FIGURA 1.4: Pubblicazioni su ES e KBS dal 1975 ad oggi. Risultati da Scopus.

È possibile identificare alcune categorie di sistemi esperti, riportati nella Tabella 1.2 con alcuni esempi d'uso riportati in letteratura.

Tipo di ES	Esempi di applicazione
Diagnostica	MYCIN: diagnosi di malattie infettive
Identificazione	DENDRAL: usato nell'identificazione di composti organici
Supporto alle decisioni	DART: assistenza nella distribuzione di risorse militari

TABELLA 1.2: Tipi di ES con relativi esempi [14].

Il caso d'uso tipico di un sistema esperto, indifferentemente dall'applicazione dello stesso, prevede che l'utente descriva il problema (fornendo quindi i fatti) e il sistema offra consigli, suggerimenti e raccomandazioni. Nel paragrafo 1.2.1.1 viene descritto nello specifico il funzionamento di un ES.

1.2.1.1 Funzionamento

Lo schema funzionale di un sistema esperto è rappresentato in Figura 1.5. Come mostrato anche in Figura 1.3, i componenti essenziali del sistema esperto sono la *knowledge base* (KB), nella quale sono memorizzate le regole, l'interfaccia con il mondo esterno, che si occupa di ricevere i fatti, ed il motore inferenziale, che seleziona la regola da applicare.

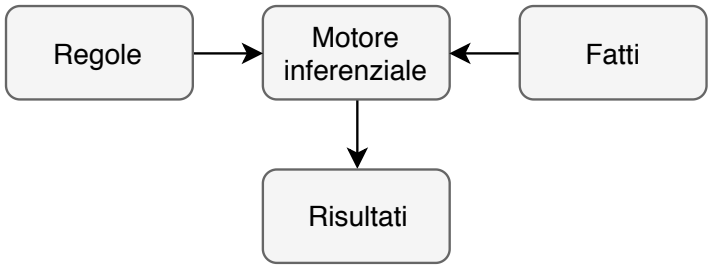


FIGURA 1.5: Funzionamento di base di un sistema esperto in modalità *forward-chaining* [16].

Il meccanismo di inferenza deduttivo cerca, tra tutte le regole presenti nella KB, quelle che possono essere attivate dai fatti ricevuti in input; cerca cioè le regole i cui termini antecedenti siano soddisfatti dai fatti forniti dall'utente. Qualora più regole possano essere applicate, il motore inferenziale risolve i conflitti[4] e ne applica una sola. Il risultato sarà quindi dato dai termini conseguenti della regola applicata.

1.2.1.2 Il bisogno della spiegazione

Un requisito ritenuto fondamentale per i sistemi esperti è la capacità di fornire la spiegazione del ragionamento che li ha portati ad una certa conclusione [17]. Il concetto di spiegazione (*explanation*) è strettamente collegato ad altri due concetti, elencati di seguito [17]:

[4] I conflitti sono risolti applicando *meta-regole*, ovvero regole che stabiliscono il comportamento del motore inferenziale. Tipicamente si può scegliere di applicare la regola con la priorità più alta, oppure quella più specifica.

- **interpretabilità**: un sistema è detto interpretabile se le sue azioni possono essere comprese da un umano, sia attraverso l'introspezione che grazie ad una spiegazione fornita dal sistema stesso;
- **giustificazione**: spiega perché la decisione presa è valida. Intuitivamente, un sistema può spiegare perché una certa decisione è valida: può inoltre accennare a *come* esattamente è arrivato a tale conclusione.

È stato dimostrato da [17] che l'interpretabilità e le giustificazioni rendono più accettabile, da parte degli utenti, il risultato generato, aumentando la loro confidenza e fiducia verso il sistema, assieme alla possibilità per l'utente di poter valutare se una predizione sia corretta o meno.

1.2.1.3 Applicazioni

Le possibili applicazioni dei sistemi esperti sono vaste. Non tutti i problemi tuttavia sono efficacemente ed efficientemente risolvibili con l'adozione di un ES; secondo [14] i sistemi esperti sono applicabili con profitto quando:

- il problema è rilevante nel business, cioè si risparmierebbero sia tempo, che denaro grazie all'uso del sistema esperto;
- la conoscenza richiesta è disponibile e stabile. Ciò significa che gli esperti umani sono disponibili a fornire senza ambiguità la conoscenza necessaria per costruire la *knowledge base*;
- la conoscenza richiesta è scarsa;
- il problema è ricorrente, cioè il sistema esperto verrà usato molte volte, in luoghi diversi, per un lungo periodo di tempo;
- il problema è adeguatamente complicato. In alcuni casi (quando il problema è semplice e la conoscenza richiesta è bassa) potrebbe essere più conveniente addestrare un numero elevato di esperti umani. D'altro canto, problemi molto complessi potrebbero essere risolti solamente da esperti umani;
- il dominio è ben definito e di dimensione gestibile, cioè il numero di regole necessarie per descriverlo è sufficientemente piccolo;
- la soluzione dipende da ragionamenti logici, non senso comune o conoscenza generale.

Siccome i sistemi esperti sono basati sulla conoscenza, essi sono intrinsecamente *rigidi*, poco flessibili. Questo perché la conoscenza nella *knowledge base* è statica e

non deriva dall'analisi dei fatti. Ciò comporta che sistemi di questo genere non possano adattarsi all'utilizzo in vari contesti, a meno di non creare un'altra KB che si adatti al nuovo contesto.

Viceversa, come si vedrà nella sezione 1.3, i sistemi di *machine learning* sono lo strumento ideale per lo sviluppo di sistemi che imparino dai dati e che siano quindi capaci di adattarsi al contesto nel quale operano.

1.2.2 *Knowledge Engineering*

L'ingegneria della conoscenza - chiamata anche *Knowledge Engineering* (KE) - è il processo di sviluppo di sistemi basati sulla conoscenza in qualsiasi campo, pubblico o privato, nel commercio o nell'industria [14]. Con conoscenza si intende l'associazione funzionale esplicita che sussiste tra dati ed informazioni [18]. Fin da subito è bene precisare che dati, informazioni e conoscenza non sono elementi statici, ma sono parti del processo che porta alla formazione della conoscenza stessa. Un esempio di questo processo viene riportato in Figura 1.6.

FIGURA 1.6: Dati, informazioni e conoscenza [14].

Da un punto di vista ingegneristico, l'esempio della Figura 1.6 si può riassumere nella seguente regola: *se* fuori fa freddo, *allora* indossa una giacca. Si nota quindi come la generica regola di produzione *if-then* risulti essere una struttura molto comoda per la rappresentazione della conoscenza.

Dalla definizione di ingegneria della conoscenza appare evidente che essa è un ramo dell'ingegneria strettamente legato ai sistemi esperti: la costruzione della *knowledge base* di un ES coinvolge necessariamente la KE.

1.2.2.1 Acquisizione della conoscenza

L'acquisizione della conoscenza, nota anche come elicitazione, è il processo che consente di esplicitare la conoscenza di dominio ricavandola da svariate fonti, che possono essere uno o più esperti di dominio, documenti scritti, programmi software, video o altro.

Tale processo coinvolge quindi l'esperto ed il *knowledge engineer*, una figura nata parallelamente allo sviluppo dei sistemi esperti. Questa figura ha il compito di estrarre quante più informazioni possibili dall'esperto, per poi trasferirle nella *knowledge base* del sistema. Il *knowledge engineer* deve quindi avere una profonda conoscenza dello strumento e deve essere in grado di condurre interviste proficue con gli esperti di dominio, il che richiede una solida conoscenza del dominio stesso.

L'acquisizione della conoscenza viene effettuata attraverso [14]:

- interviste agli esperti umani;
- altre fonti di conoscenza, quali questionari, manuali, casi di studio e libri di testo;
- *data mining*, qualora vi fossero grandi quantità di dati disponibili.

Questi approcci, seppur validi in linea teorica, presentano dei problemi quando vengono applicati; in particolare si presenta il cosiddetto *knowledge acquisition bottleneck*. Questo problema si verifica essenzialmente quando lo sviluppatore del sistema non possiede una conoscenza del dominio sufficiente.

Il problema dell'acquisizione della conoscenza si può riassumere in quattro punti [19]:

- lentezza di acquisizione: i processi che permettono di acquisire e formalizzare la conoscenza sono generalmente lenti e dispendiosi in termini di tempo;
- latenza di acquisizione: il processo di acquisizione è lento ed è spesso accompagnato da un ritardo tra quando la conoscenza è creata e quando diviene disponibile per essere condivisa;
- inaccuratezza della conoscenza: spesso l'acquisizione della conoscenza da esperti introduce degli errori, dovuti ad una scarsa efficacia comunicazionale, ad errori interpretativi o ad errori degli esperti. Anche i processi di *data mining* possono essere inaccurati, evidenziando talvolta relazioni spurie tra i dati;
- trappola della manutenzione: con l'accrescimento della conoscenza nella *knowledge base*, cresce anche il bisogno di manutenerla.

L'esistenza del collo di bottiglia nella fase di acquisizione della conoscenza ha smorzato i primi entusiasmi ottenuti con i sistemi esperti nei primi anni seguenti alla loro nascita.

Una soluzione al citato problema è offerto da sistemi che siano in grado di apprendere autonomamente [13]. Euristicamente, nei casi in cui sia difficile l'elicitazione della conoscenza relativa ad uno specifico dominio, sarebbe interessante riuscire ad automatizzare questo processo, in modo che un sistema possa derivare una *knowledge base* da una serie di esempi. Si possono distinguere due categorie di sistemi di questo tipo [13]:

- apprendimento simbolico: il sistema è in grado di formulare e modificare la *knowledge base*;
- apprendimento numerico: il sistema utilizza dei modelli numerici per adattarsi agli esempi (*machine learning*).

1.2.2.2 Rappresentazione della conoscenza

Come anticipato in precedenza, la rappresentazione della conoscenza avviene attraverso regole *if-then* [13, 14]:

$$IF < situazione > THEN < azione >$$

I termini antecedenti possono essere collegati tra loro da proposizioni logiche (AND, OR, NOT, ELSE, ecc.) in modo da formare condizioni arbitrariamente complesse.

1.2.2.3 Valutazione del KBS

La valutazione è il processo con il quale si asserisce il valore di un KBS. Ciò significa controllare non solo che il livello di prestazioni del sistema sia accettabile, ma anche che esso sia usabile, efficiente e redditizio.

Tale processo comprende le seguenti fasi [14]:

- **validazione**, nella quale si misura la prestazione del KBS. Si compara l'output del sistema con quello che sarebbe l'output di un esperto; si vuole che il sistema operi ad un accettabile livello di accuratezza;
- **verifica**, nella quale ci si accerta che il KBS sia stato realizzato correttamente. Si controlla quindi che le regole siano logiche e consistenti con la conoscenza acquisita durante la prima fase della *knowledge engineering*.

La necessità di valutare il KBS deriva dal fatto che nei processi di acquisizione, trasferimento e rappresentazione della conoscenza vi possano essere delle distorsioni.

Validazione

La fase di validazione si occupa, in termini generali, di testare la corrispondenza tra dominio di interesse e dominio della conoscenza trasferita al KBS. Più specificamente, tale fase consiste nella definizione dei casi di prova da usare (e quanti usarne). L'applicazione del KBS sui casi di test consente di stabilire se i risultati di quest'ultimo sono conformi a quelli - già predetti - degli esperti.

Verifica

La fase di verifica è particolarmente importante, perché consente di prevenire situazioni dove le regole non sono consistenti. Si prenda in considerazione l'esempio seguente:

	Regola
1	IF A AND B AND C THEN X
2	IF A AND B THEN X

In questo caso la regola 1 è sussunta nella regola 2: infatti la 1 è più specifica dell'ultima. Siccome inoltre il termine conseguente è lo stesso, la regola 1 è non necessaria.

Le seguenti regole invece sono lecite:

	Regola
1	IF A AND B AND C THEN Y
2	IF A AND B THEN X

in quanto hanno termini conseguenti diversi. Sarà compito del motore inferenziale risolvere i conflitti che possono derivare da situazioni di questo tipo. Infatti se la regola 1 è verificata, allora lo è necessariamente anche la regola 2. Devono esistere delle meta-regole che stabiliscano quale debba essere applicata[5].

Un altro esempio è quello che si verifica quando si utilizzano termini antecedenti non necessari. Si consideri l'esempio seguente:

[5] Alcune soluzioni che si possono adottare sono: applicare la prima regola che si incontra, oppure applicare la regola più specifica.

	Regola
1	IF D AND E THEN X
2	IF D AND F THEN Y

In questo esempio la condizione D è superflua, in quanto non è discriminante nella determinazione del termine conseguente; per questo motivo, va eliminata.

1.3 *Machine Learning*

L'apprendimento automatico (noto ai più col termine di *Machine Learning*) è la scienza che si occupa di programmare i computer in modo che siano capaci di apprendere dai dati [20]. Una definizione più ingegneristica è la seguente: «Un programma apprende dall'esperienza E rispetto ad un certo compito C ed alle prestazioni P se le sue prestazioni nello svolgere C, misurate da P, aumentano con l'esperienza E». La definizione introduce tre elementi che rivestono un ruolo fondamentale nel *machine learning* [20]:

- il compito C, ovvero il problema da risolvere;
- l'esperienza E, rappresentata dagli esempi che vengono presentati al sistema e viene comunemente chiamato set di addestramento - *training set*. Ogni istanza del set costituisce un *esempio*;
- la misura delle prestazioni P, ovvero una quantità numerica ben definita che permetta di stabilire se il sistema sta performando bene o meno.

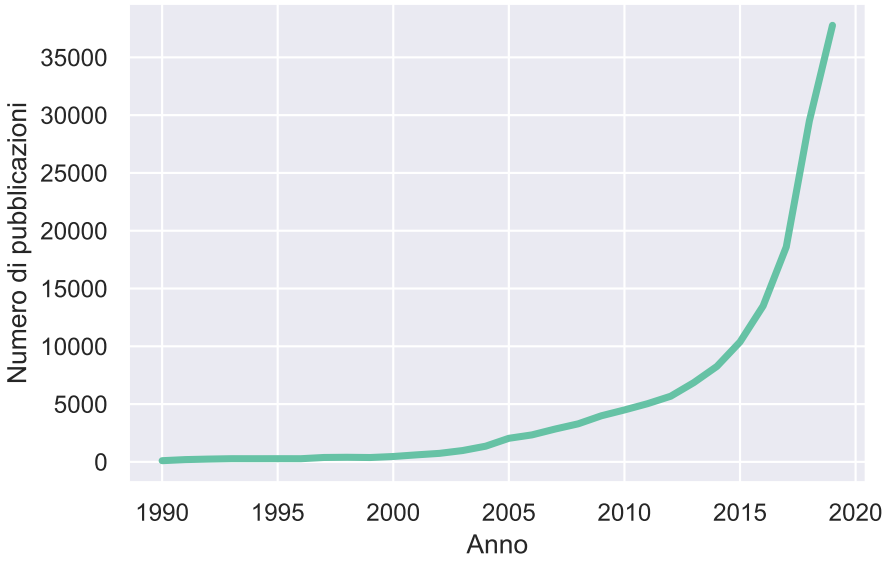


FIGURA 1.7: Pubblicazioni sul *Machine Learning* dal 1984 ad oggi. Risultati da Scopus.

Appare chiara dunque la differenza di questo approccio con quanto accade nei sistemi basati sulla conoscenza; nei primi la conoscenza non viene esplicitata, formalizzata e conferita al programma sotto forma di regole, ma è il programma stesso (attraverso algoritmi e tecniche di apprendimento automatico) a ricavare delle regole *implicite* derivanti dai dati al fine di risolvere un problema. Tipicamente inoltre queste regole non sono accessibili ed interpretabili dall'utilizzatore.

La ricerca in ambito *machine learning* (a differenza di quella in ambito ES) è molto attiva, come si evince dalla Figura 1.7; si può affermare inoltre che proprio in questi ultimi anni si stia assistendo all'esplosione dell'interesse scientifico in questo campo.

1.3.1 Idea di base

Il funzionamento di base dei sistemi di *machine learning* può essere diviso in due macro-fasi [20]:

- **addestramento**: al modello vengono forniti degli esempi (dei quali la predizione desiderata può essere nota o meno), che rappresentano al meglio il problema in esame. Il sistema usa questi dati per adattarsi al problema;

- **applicazione**: consiste nell'applicazione del sistema addestrato su dati *nuovi*, non usati durante l'addestramento. Le predizioni saranno tanto più ac-

curate quanto migliore è stata la fase di training[6].

Il successo degli algoritmi di apprendimento automatico è strettamente legato alla disponibilità di enormi quantità di dati da usare nella fase di addestramento. Non c'è da sorprendersi quindi che in questo periodo storico si stia assistendo ad una crescita di tali sistemi, in quanto vi è sia la disponibilità di dati, sia la potenza di calcolo necessaria a processarli.

1.3.2 Tipi di *machine learning*

I sistemi di apprendimento automatico sono numerosi. Secondo [20] i criteri secondo i quali gli algoritmi di *machine learning* possono essere classificati sono:

- se gli algoritmi richiedono, durante l'addestramento, la supervisione di un umano o meno. Si distinguono dunque apprendimenti di tipo:
  - **supervisionato**: i dati di addestramento includono anche la soluzione desiderata;
  - **non supervisionato**: i dati di addestramento non includono la soluzione desiderata;
- se gli algoritmi sono capaci di apprendere in modo incrementale. Si distinguono:
  - apprendimento **incrementale**: il sistema può essere addestrato in modo incrementale, sottoponendogli esempi uno alla volta o in piccoli lotti;
  - apprendimento **a lotti**: il sistema viene addestrato utilizzando tutti i dati a disposizione. Il programma dunque viene prima addestrato e solamente poi eseguito;
- se la predizione avviene tramite confronto di nuovi dati con quelli noti o tramite costruzione di un modello sulla base dei dati di addestramento. Si distinguono quindi:
  - apprendimento **basato sugli esempi**: il sistema fornisce predizioni sulla base del grado di similarità dell'esempio con quanto contenuto nel set di addestramento;
  - apprendimento **basato sui modelli**: il sistema costruisce un modello sulla base degli esempi contenuti nel set di addestramento. Le predizioni vengono effettuate utilizzando il suddetto modello.

[6] La fase di training non è stata efficace se il sistema presenta problemi di *overfitting* o *underfitting*.

Bisogna specificare che queste categorie non sono esclusive: uno stesso algoritmo può appartenere a più di una categoria.

Nell'ambito dei sistemi di apprendimento supervisionato, si distinguono algoritmi di [20]:

- *classificazione*, quando l'output dell'algoritmo è la classe di appartenenza dell'esempio fornito in input. Si ha quindi in uscita un valore discontinuo (numerico o non);
- *regressione*, quando l'output del sistema è un insieme continuo. In uscita si ha quindi un valore numerico.

Nel caso di sistemi di apprendimento non supervisionato invece si parla di *clustering*, ovvero l'identificazione di gruppi di esempi che sono simili tra loro. È onere di questi algoritmi trovare le relazioni tra gli attributi degli esempi che caratterizzano i *clusters*.

1.3.3 Applicazioni e possibili problemi

I sistemi di *machine learning* trovano naturale applicazione in una serie di problemi. La loro descrizione è riportata di seguito [20]:

- la soluzione classica al problema da risolvere richiederebbe una grande quantità di aggiustamenti manuali o un enorme set di regole;
- problemi per i quali non è possibile trovare una soluzione con l'approccio tradizionale;
- problemi nei quali il dominio è in evoluzione - gli algoritmi di *machine learning* possono adattarsi;
- problemi nei quali sia disponibile una grande quantità di dati.

Da quanto visto fino ad ora, i sistemi di *machine learning* sono essenzialmente basati su di un algoritmo di apprendimento e sui dati utilizzati nell'addestramento. Due sono quindi le cause per l'eventuale fallimento di un sistema di apprendimento automatico [20]:

- problemi di **dati**. In particolare, una o più delle seguenti situazioni si può verificare:
  - la quantità di dati di addestramento *non è sufficiente*;
  - i dati *non sono rappresentativi* del problema, cioè i dati di esempio non sono sufficientemente generali, nel senso che non descrivono esaustivamente il problema;

  - la qualità dei dati è *scarsa*, cioè vi sono degli *outliers* nel set di addestramento che compromettono la fase di training;
  - i dati comprendono degli attributi che sono *irrilevanti*, o non prendono in cosiderazione attributi rilevanti. Il processo di selezione degli attributi è fondamentale, e prende il nome di *feature engineering;*

- problemi di **algoritmi**:
  - *overfitting*: tale problema si verifica quando l'algoritmo è troppo complesso in relazione al problema che si sta tentando di risolvere. Si è in presenza di overfitting quando il sistema ha ottime prestazioni sui dati di addestramento, ma pessime sugli esempi di test;
  - *underfitting*: come intuibile, si ha underfitting quando il modello è troppo semplice per apprendere la struttura sottostante al problema che si sta tentando di risolvere.

1.4 Gestione delle incertezze

I modelli analizzati fino ad ora hanno senso solamente se si assume che ogni ipotesi può essere vera, falsa o non nota. Inoltre, sfruttando l'ipotesi del mondo chiuso citata in precedenza, si ha che ogni condizione non nota è assunta essere falsa; quindi gli stati possibili per ogni ipotesi sono solo vero o falso.

Questo modello decisionale dicotomico il più delle volte si discosta molto dai problemi reali, i quali sono affetti da incertezza. Se ne possono distinguere tre tipi [13]:

- incertezza della regola stessa: ogni regola presente nella *knowledge base* ha un intrinseco livello di certezza;
- incertezza nella condizione: potrebbe esserci un'incertezza nella veridicità della condizione antecedente della regola, che quindi si propaga al termine conseguente;
- uso del linguaggio vago: spesso nelle regole si fa riferimento a termini linguistici incerti, come «alto», «basso», «poco», ecc. che hanno bisogno di essere mappati per assumere un significato preciso. Inoltre il significato di questi termini potrebbe variare passando da un contesto all'altro.

È importante distinguere queste fonti di incertezza in quanto vanno trattate diversamente. I primi due tipi vengono gestiti attraverso la teoria *probabilistica* delle reti Bayesiane, mentre l'ultima viene affrontata con la teoria *possibilistica* della logica fuzzy [13].

Attenendosi allo scopo della tesi, si ritiene sufficiente dare una descrizione delle logiche sfumate - o fuzzy - nel prossimo paragrafo.

1.4.1 Logica sfumata

La teoria della logica sfumata è stata sviluppata da Zadeh nel 1965 per operare matematicamente con termini linguistici. Una delle caratteristiche chiave dei sistemi fuzzy risiede nell'abilità nel trattare l'incertezza che affligge i sistemi fisici.

Si distinguono due logiche [13]:

- crisp: sono gli usuali sistemi dicotomici, nei quali una condizione può assumere gli stati di vero o falso;
- fuzzy: permettono di definire un *grado di appartenenza* di una condizione ad un certo insieme. Una certa condizione può avere gradi di appartenenza 0.75 e 0.2 agli insiemi VERO e FALSO, rispettivamente.

Le logiche sfumate sono alla base del ragionamento umano e dei processi decisionali; l'adozione di queste logiche permette quindi [13]:

- un'efficace rappresentazione della conoscenza dei fatti di un problema;
- un'efficiente elaborazione della stessa grazie ad una struttura numerica sottostante alla rappresentazione linguistica.

È stato dimostrato che la logica fuzzy può essere convenientemente applicata nei sistemi esperti e in altre applicazioni AI [21].

Si passa ora a descrivere brevemente il funzionamento dei sistemi fuzzy.

1.4.2 Insiemi fuzzy

I fuzzy sets sono un'estensione dei crisp sets, nei quali ci si svincola dal concetto di appartenenza o non appartenenza. Gli insiemi a logica sfumata infatti permettono di avere un'appartenenza parziale, da 0 ad 1, di una condizione ad un certo insieme. L'appartenenza di un elemento x al fuzzy set A viene descritta da una *membership function* definita come:

$$\mu_A(x) : X \to [0;1]$$

Il fuzzy set A può essere convenientemente rappresentato con un'etichetta linguistica: classici esempi sono *small*, *medium*, *large*, ecc.

Si osservi la Figura 1.8: qui sono stati definiti tre fuzzy sets (*low*, *medium*, *high*) per la variabile *Distance*, il cui dominio è $[0;100]$.

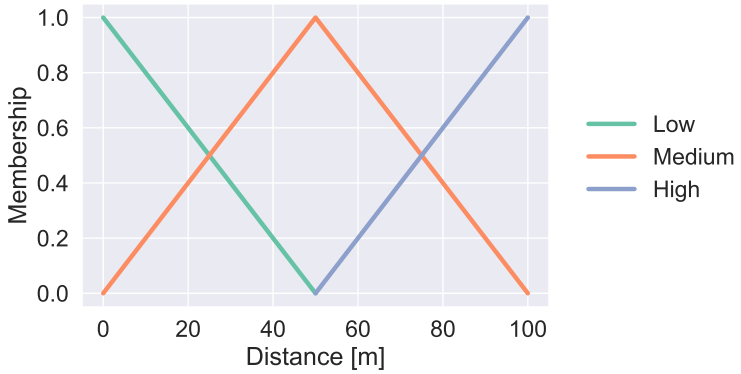


FIGURA 1.8: Fuzzy sets e gradi di appartenenza.

Ad esempio, una distanza $d = 40\ m$ avrà i seguenti gradi di appartenenza:

- $\mu_{LOW} = 0.2$;
- $\mu_{MEDIUM} = 0.8$;
- $\mu_{HIGH} = 0$.

Si ritiene opportuno precisare che la definizione delle *membership functions* è a discrezione di chi sviluppa il sistema fuzzy. In letteratura sono riportati esempi di funzioni di appartenenza comunemente utilizzate e *best practices* da seguire nella progettazione; non esistono comunque regole prestabilite. Quello che è certo è che la definizione delle stesse può essere più o meno conveniente ed efficace per la risoluzione di un problema. L'abilità nella costruzione di un sistema fuzzy sta quindi nella scelta opportuna del numero e del tipo di funzione di appartenenza, oltre che alla definizione delle regole (discusse nel paragrafo seguente).

1.4.3 Regole fuzzy

Una tipica regola fuzzy è composta da [13]:

- condizioni antecedenti;
- conseguente.

Questa struttura può essere convenientemente rappresentata nel costrutto *if-else*:

$$IF(x\ is\ A)\ AND\ (y\ is\ B)\ THEN\ (z\ is\ C)$$

In tale regola si ha che ogni condizione antecedente associa una variabile ad un fuzzy set.

Si fa notare che tale struttura è la stessa che può essere usata per la rappresentazione della conoscenza nei sistemi esperti; il fatto che gli antecedenti si riferiscano a dei fuzzy sets o meno non cambia la struttura della regola stessa.

1.4.4 Meccanismo inferenziale

Il meccanismo inferenziale di un sistema fuzzy è il processo attraverso il quale si ottiene un valore numerico in uscita partendo da n ingressi numerici.

Si supponga di avere un sistema composto da:

- n variabili di ingresso: $x_i \quad i = 1, ..., n$;
- p fuzzy sets per ciascuna variabile di ingresso: $A_{k,i} \quad k = 1, ..., p$;
- p fuzzy sets per la variabile di output: $B_k \quad k = 1, ..., p$;
- m regole.

I processi che costituiscono il meccanismo inferenziale fuzzy sono descritti di seguito [22]:

1. fuzzificazione degli ingressi. Per ogni variabile di input si determina il grado di appartenenza della stessa ai fuzzy sets:

$$\mu_{A_k}(x_i) = \mu_{k,i}$$

2. grado di attivazione delle regole. Per ogni regola si calcola il grado di attivazione secondo la seguente:

$$\lambda_j = \min_i (\mu_{k,i}), \quad j = 1, ..., m$$

3. effetto sul set conseguente. A seconda del grado di attivazione della regola, l'effetto sul set conseguente B_k genera un nuovo set B'_j che viene calcolato in due modi diversi:

$$\mu_{B'_j}(y) = \begin{cases} \lambda_j \cdot \mu_{B_k}(y) & \textit{correlation product} \\ \lambda_j \wedge \mu_{B_k}(y) & \textit{correlation minimum} \end{cases}$$

4. sovrapposizione di tutte le regole:

$$\mu_B(y) = \sum_{j=1}^{m} \mu_{B'_j}(y)$$

5. defuzzificazione con il metodo del centroide:

$$y = \frac{\int y \cdot \mu_B(y) dy}{\int \mu_B(y) dy}$$

La potenza del meccanismo inferenziale dei sistemi fuzzy rispetto ai sistemi tradizionali risiede nel fatto che vi sono tante regole variamente soddisfatte, le quali contribuiscono a determinare l'output. Non vi è dunque una sola regola che determina l'output. Per questa caratteristica i sistemi fuzzy sono molto usati nei sistemi di controllo, in quanto permettono una variazione lenta (non crisp) della variabile di uscita.

Tale caratteristica ha lo svantaggio principale che, in fase di debugging, è più difficile risalire al motivo per cui si è ottenuto un certo risultato.

1.4.5 Insiemi crisp

A differenza degli insiemi fuzzy, gli insiemi crisp sono insiemi non sovrapposti e hanno una forma rettangolare. Ciò significa che il grado di appartenenza di una variabile ad un insieme crisp può essere 0 o 1. La Figura 1.9 esemplifica tale concetto.

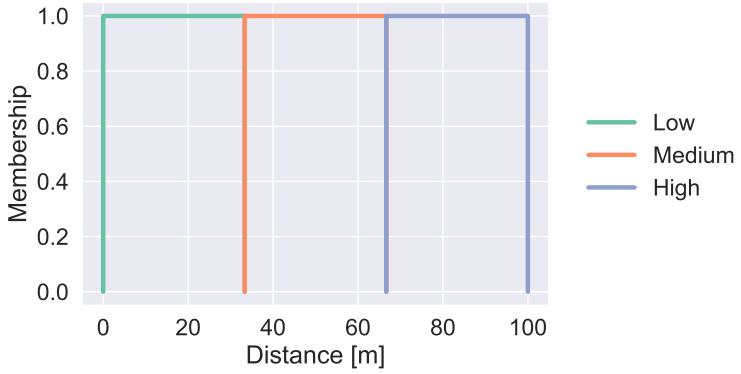


FIGURA 1.9: Crisp sets e gradi di appartenenza.

Gli insiemi crisp possono quindi essere studiati come un caso particolare di insiemi fuzzy.

L'uso di questi insiemi in un sistema decisionale consente quindi di maneggiare variabili affette da incertezza (cioè descritte da etichette linguistiche) senza introdurre la complessità del meccanismo inferenziale fuzzy, nel quale più regole possono contemporaneamente attivarsi. Se l'insieme di regole è valido, allora un certo set di fatti attiverà una ed una sola regola[7].

Chiaramente l'output del sistema sarà meno preciso di quanto si otterrebbe con l'uso di insiemi fuzzy e meno sensibile alle piccole variazioni del valore delle variabili; il vantaggio è che il sistema ne guadagna in chiarezza procedurale, rendendo più agevole la fase di debugging.

[7]Se il set di regole è meccanicamente perfetto

1.5 Agenti intelligenti

Lo sviluppo di sistemi complessi richiede solitamente la decomposizione del problema in componenti più elementari. Ciascuna di queste componenti viene poi gestita grazie ad un sistema progettato per risolvere un problema specifico: questi sono detti **agenti intelligenti**, i quali sono sviluppati con una delle tecniche fino ad ora analizzate per raggiungere un obiettivo specifico.

Gli agenti intelligenti sono dei sistemi che, all'interno di un certo ambiente, sono in grado di prendere autonomamente delle decisioni al fine di raggiungere gli obiettivi prefissati.

Le caratteristiche chiave di un agente intelligente sono [13]:

- **autonomia**: la capacità dell'agente di prendere autonomamente le decisioni sulla base del contesto e dell'esperienza;
- **persistenza**: il funzionamento continuo dell'agente all'interno dell'ambiente per il quale è stato sviluppato;
- **collocazione**: la capacità di reazione alle richieste dell'ambiente di lavoro.

In aggiunta, altre caratteristiche che un agente può possedere sono [13]:

- **reattività**, quando le azioni sono una diretta conseguenza di un fatto;
- **adattabilità** alla variazione dell'ambiente di lavoro;
- **capacità sociali**, ovvero la capacità di un agente di cooperare con altre entità (agenti o umani)

L'ultima caratteristica discussa costituisce la base per i sistemi multi-agenti, analizzati nel prossimo paragrafo.

1.5.1 Sistemi multi-agente

Gli agenti intelligenti risultano essere interessanti soprattutto quando vi è la possibilità di farli interagire tra loro e farli operare assieme; i sistemi di questo tipo sono detti multi-agente. I benefici che i sistemi multi-agente comportano sono elencati di seguito [13]:

- velocità ed efficienza del sistema, in quanto gli agenti possono funzionare contemporaneamente e comunicare in modo asincrono;
- robustezza ed affidabilità, in quanto nessun agente è fondamentale. Ciò significa che le prestazioni dell'intero sistema decadono dolcemente se i singoli agenti dovessero fallire;

- scalabilità generalmente garantita tramite semplice aggiunta di nuovi componenti al sistema;
- costo generalmente più basso di sviluppo di tanti piccoli agenti rispetto ad un unico grande sistema centralizzato;
- manutenzione semplificata grazie alla modularità del sistema.

La realizzazione di un sistema multi-agente è strettamente legata alla sua architettura; la ricerca in questo ambito ha portato allo sviluppo della cosiddetta architettura *blackboard*, una memoria centrale accessibile da tutti gli agenti nella quale viene rappresentato lo stato corrente della conoscenza. Lo schema di questa architettura è riportato in Figura 1.10.

Il termine *blackboard* esemplifica bene l'analogia che permette di capire come funzionano i sistemi multi-agente: si pensi ad un team di esperti, ognuno specializzato in una certa area, che scrivono nuove informazioni su di una lavagna a seconda di quanto vi è già scritto.

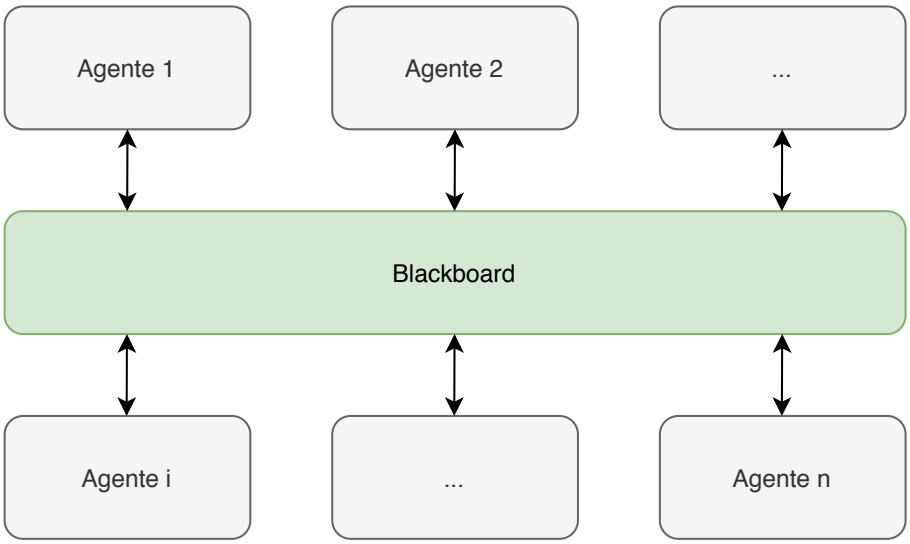


FIGURA 1.10: Architettura di un sistema *blackboard*. Gli *n* agenti si interfacciano con la memoria centrale, creando il sistema multi-agente [13].

La prima pubblicazione relativa all'architettura *blackboard* risale al 1975 [23] ed è stata successivamente riconosciuta da [24] come la tecnologia chiave per la realizzazione di sistemi multi-agente.

1.6 Stato dell'arte

Come mostrato nei paragrafi precedenti, la ricerca in ambito di intelligenza artificiale, sistemi esperti e *machine learning* è attiva (soprattutto per quest'ultimo).

Per quanto riguarda i sistemi esperti, sono numerosi gli articoli in letteratura dove si utilizzano i sistemi fuzzy per gestire le incertezze nel processo decisionale [25, 26], che dimostrano le potenzialità delle logiche sfumate in ambito *decision-making*.

Quello che sorprende però è che vi siano pochi studi in letteratura che coniughino sistemi esperti e *machine learning*. Sebbene sia stato ipotizzato alla fine del secolo scorso che l'integrazione tra i due sistemi potesse portare notevoli benefici [27], sono rare le applicazioni in tale senso [28]. Al meglio della conoscenza, [28] rappresenta lo stato dell'arte per quanto riguarda l'integrazione tra *knowledge-based systems* e *machine learning* basato sul feedback dell'utente.

Tale approccio consiste in un classificatore fuzzy basato sulla conoscenza sotto forma di regole in grado di adattarsi sulla base dei feedback che riceve dall'utente. Nel seguito di questo paragrafo viene descritto il sistema proposto da [28].

1.6.1 Motivazioni

La scelta del sistema classificatore (basato sulle regole o sui dati) non può essere presa senza considerare alcuni aspetti [28]:

- sebbene gli algoritmi di *machine learning* offrano spesso un'accuratezza più elevata, questa viene raggiunta con il costo di un maggior onere computazionale, nell'ipotesi che vi sia una grande quantità di dati disponibile;

- i modelli basati sulla conoscenza di dominio permettono di scavalcare il bisogno di dati. Il sistema però soffrirà nell'adattarsi a distribuzioni reali di dati.

La soluzione che permette di superare i limiti di entrambi i sistemi è quella di sviluppare un sistema ibrido, che integri la conoscenza di dominio (sotto forma di regole) con l'adattabilità del sistema ai dati reali (quindi con algoritmi di apprendimento automatico).

Infatti un sistema basato sulle regole può predire correttamente i casi tipici, ma si trova in difficoltà quando deve gestire casi atipici; per risolvere questo problema, si usano i modelli di *machine learning*. Comunque, tali modelli hanno difficoltà a classificare gli esempi quando i dati disponibili per l'addestramento sono pochi o non rappresentativi.

I modelli ibridi si prefiggono di integrare gli aspetti positivi di entrambi i metodi.

1.6.2 Architettura

Nel sistema proposto la classificazione viene così perseguita [28]:

- la conoscenza degli esperti viene trasformata nella *knowledge base* sotto forma di sistema basato su regole fuzzy, costruendo un set iniziale di regole;
- le membership functions dei fuzzy sets sono ottimizzate integrando la conoscenza di dominio con l'esperienza derivata dai dati. Si hanno quindi due basi di conoscenza, quella dell'esperto e quella appresa dall'analisi dei dati. Il peso relativo di queste viene controllato da un parametro conservativo, che influenza il comportamento del sistema rispetto alla conoscenza dell'esperto e quella appresa.

Rispetto ai classici sistemi basati sulla conoscenza, l'uso di sistemi ibridi migliora l'accuratezza di classificazione, specialmente quando la conoscenza dell'esperto è incompleta o imprecisa.

Nella Figura 1.11 viene descritta l'architettura del sistema.

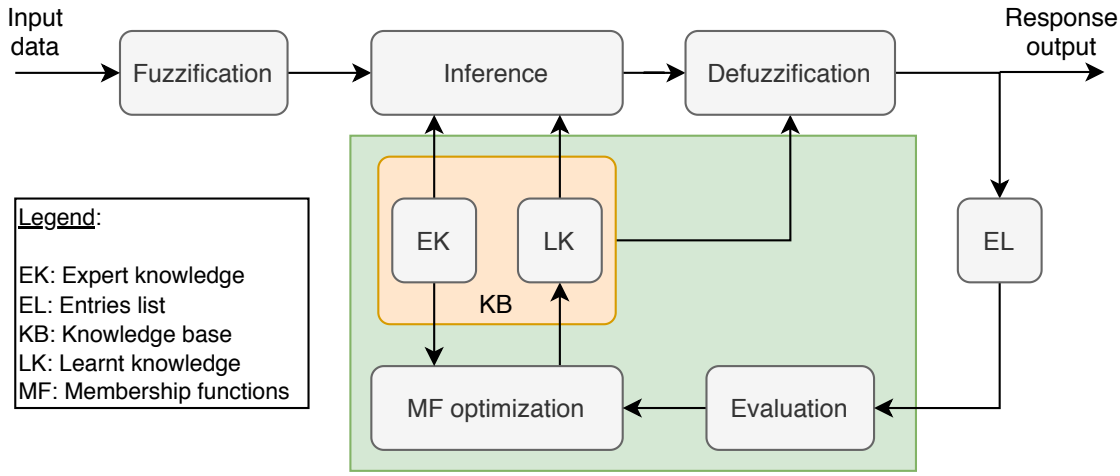


FIGURA 1.11: Architettura del sistema ibrido adattivo basato su regole fuzzy [28].

1.6.3 Meccanismo di aggiornamento

Il meccanismo che consente l'apprendimento di nuova conoscenza (o meglio, l'adattamento di quella esistente) consiste nell'aggiornare le membership functions, mantenendo invariate le regole fornite dall'esperto. Le funzioni di appartenenza sono aggiornate con un semplice algoritmo di Cauchy. Detto σ_c il parametro che definisce la forma della membership function, si ha [28]:

$$\Delta_c = -\eta \frac{\partial E}{\partial \sigma_c}$$

con η learning rate e E funzione di errore definita di seguito [28]:

$$E = \beta E_K + (1 - \beta) E_D$$

con $\beta \in [0, 1]$ parametro conservativo[8], E_K e E_D errori quadratici medi calcolati rispettivamente sulle classi definite originariamente dall'esperto e sulle classi attuali considerando la serie di dati storici. Il meccanismo di aggiornamento esposto modifica le membership functions in modo tale da minimizzare l'errore commesso dal classificatore, sia sull'esempio proposto che su quelli già in memoria.

1.6.4 Risultati

Il confronto del sistema ibrido presentato in [28] con un classico classificatore basato su regole fuzzy, applicati a dataset UCI con diversi numeri di classi ed attributi ha mostrato che il sistema ibrido ha prestazioni migliori (accuratezza migliore) del classificatore fuzzy, in quanto riduce l'ambiguità della predizione.

Da precisare che dall'articolo non emerge un chiaro valore ottimale per il parametro conservativo β; infatti le prestazioni del sistema ibrido dipendono fortemente da questo e il valore ottimo varia a seconda del database sul quale viene applicato il sistema di classificazione.

[8] Tanto più grande è β, maggiore sarà la tendenza del sistema a ricordare la conoscenza iniziale fornita dall'esperto.

Capitolo 2

Metodologia applicata

In questo capitolo viene esposta la metodologia applicata per lo sviluppo del sistema ibrido oggetto della tesi. Innanzitutto si presentano le caratteristiche dei problemi che il sistema deve essere in grado di affrontare. Successivamente si analizzano i requisiti essenziali dello strumento e si studiano, tra le soluzioni discusse nel capitolo precedente, quelle che possano soddisfare tali esigenze. Infine si descrive dettagliatamente l'architettura e il funzionamento del sistema sviluppato, presentandone le potenzialità e i limiti.

2.1 Caratteristiche del problema

La scelta della direzione da intraprendere nello sviluppo del sistema e la definizione della sua architettura non possono essere effettuate senza considerare i problemi che il prototipo deve essere in grado di risolvere. L'analisi dei problemi deriva dallo studio dell'applicazione finale prevista per il sistema, ovvero l'integrazione all'interno di *CyberPlan* (Appendice B) per svolgere la funzione di assistente alla produzione. Di seguito sono riportate le caratteristiche comuni di questi problemi:

- natura decisionale[1];
- svolti ripetutamente;
- necessitano di una conoscenza di dominio definita e limitata;
- non sono complessi dal punto di vista logico, sebbene alcuni ragionamenti possano essere dipendenti dal contesto del problema stesso.

[1] I problemi di natura decisionale possono essere considerati come problemi di classificazione, nei quali la classe di appartenenza rappresenta l'azione da prendere.

2.2 Requisiti del sistema

I requisiti del sistema sono definiti in base alle caratteristiche definite nel paragrafo 2.1 e alla destinazione di utilizzo finale prevista. In particolare, il sistema deve:

- essere in grado di usare la conoscenza disponibile sotto forma di regole, poiché deve essere capace di effettuare previsioni senza bisogno di avere dei dati di addestramento;
- essere in grado di giustificare un certo risultato che viene proposto;
- essere adattivo, ovvero in grado di aumentare le proprie prestazioni grazie al feedback dell'utente;
- non avere limiti strutturali e/o applicativi, cioè essere in grado di risolvere problemi decisionali arbitrariamente complessi.

2.2.1 Vincoli di sviluppo

L'obiettivo di questo paragrafo è quello di presentare il contesto nel quale la soluzione è stata sviluppata, mettendo in risalto il *trade-off* che si presenta tra rapidità di sviluppo e possibilità di applicazione e verifica su casi reali. Trattandosi di un sistema da integrare all'interno di un prodotto commerciale, alcune scelte di architettura sono vincolate. In particolare:

- lo sviluppo viene effettuato all'interno di *CyberPlan*. Questa scelta è stata presa al fine di poter testare il prototipo su casi reali, senza dover successivamente replicare quanto fatto in un diverso ambiente. Ovviamente lo svantaggio di questa scelta è che non si ha la flessibilità e la rapidità di prototipazione di uno strumento di lavoro *general-purpose*, come possono esserlo ad esempio i *Jupyter Notebooks* [29];
- il sistema deve avere la struttura di un database relazionale[2], in quanto questa è la struttura dell'ambiente di sviluppo;
- il linguaggio di programmazione usato[3] è il linguaggio proprietario *cws*, che integra la funzionalità di *javascript* con la possibilità di interrogare il database attraverso *query SQL*.

[2]Un database relazionale è costituito da un numero più o meno elevato di tabelle, che possono essere legate tra loro attraverso una o più colonne.

[3]Le componenti del sistema di basso livello sono state scritte in C++ da [30].

2.3 Descrizione del sistema

I requisiti del sistema analizzati nella sezione 2.2 permettono di definire le caratteristiche del sistema. In particolare:

- deve essere un **sistema esperto**, in modo tale da poter sfruttare la conoscenza di dominio;
- la conoscenza deve essere espressa sotto forma di **regole linguistiche**, così da poter gestire l'incertezza e potersi adattare ai diversi contesti;
- le incertezze vengono gestite con **insiemi crisp**, in modo da non introdurre la complessità del meccanismo inferenziale fuzzy;
- deve esistere una componente che si occupi della **mappatura** delle regole linguistiche;
- deve migliorare le sue prestazioni sulla base di **feedback** ricevuti dall'utente. Si tratta quindi di un sistema capace di apprendere in modo incrementale;
- deve essere un sistema **multi-agente**, che permetta l'interazione tra le varie componenti;
- deve esistere un modulo di **interfaccia** con l'utente, che si occupi di ricevere i fatti noti e restituire quelli derivati.

Tali scelte di progetto hanno portato alla definizione dell'architettura del sistema riportata in Figura 2.1. Come si evince, le componenti principali del sistema sono:

- sistema di **tabelle decisionali**: consente la rappresentazione della conoscenza sotto forma di regole linguistiche;
- modulo di **mappatura**: consente la mappatura delle variabili numeriche in insiemi crisp;
- **motore inferenziale**: permette l'applicazione delle regole sui fatti noti;
- modulo di **interfaccia**: permette di ricevere in input i fatti noti e restituire in output quelli derivati;
- modulo di **apprendimento**: consente di ricevere feedback dell'utente sulle predizioni effettuate e quindi di adattarsi.

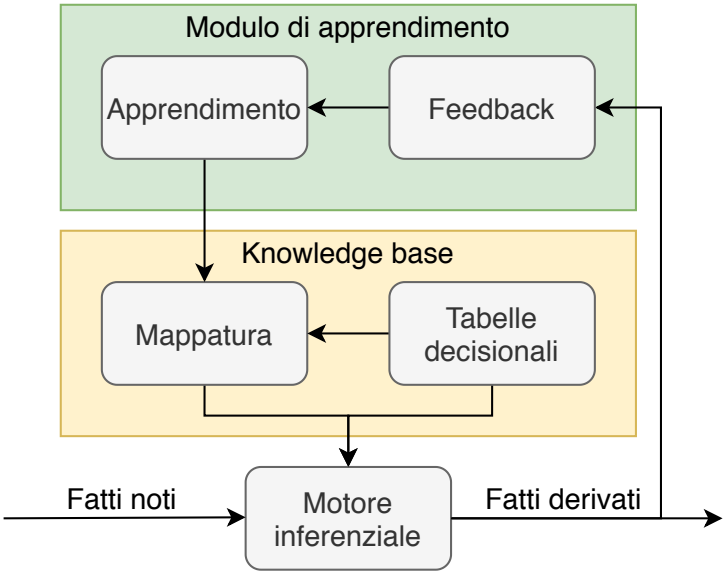


FIGURA 2.1: Architettura del sistema sviluppato. Sono evidenziate le relazioni tra le varie componenti.

Nei paragrafi seguenti vengono analizzate nello specifico le singole componenti.

2.3.1 Tabelle decisionali

Il sistema sviluppato consente la creazione della *knowledge base* tramite definizione di regole linguistiche. Per poter fare ciò, è stato necessario definire alcuni elementi nella struttura del database:

- *rulebase*: è il contenitore di tutte le regole che rappresentano la conoscenza di un dominio specifico e che permettono la risoluzione di un determinato problema;
- *attributo*: è una variabile che assume un ruolo discriminatorio nella logica di una tabella decisionale. Gli attributi sono dunque tutti i termini - antecedenti e conseguenti - presenti nelle regole;
- *valore*: è un'etichetta linguistica legata all'attributo. Ogni valore definisce un insieme crisp per l'attributo al quale si riferisce;
- *regola*: è creata dalla concatenazione (con operatore logico AND) di valori relativi ad attributi differenti.

Data la struttura di database relazionale del sistema, gli elementi appena descritti trovano definizione in apposite tabelle. La Figura 2.2 descrive le relazioni che sussistono tra di esse.

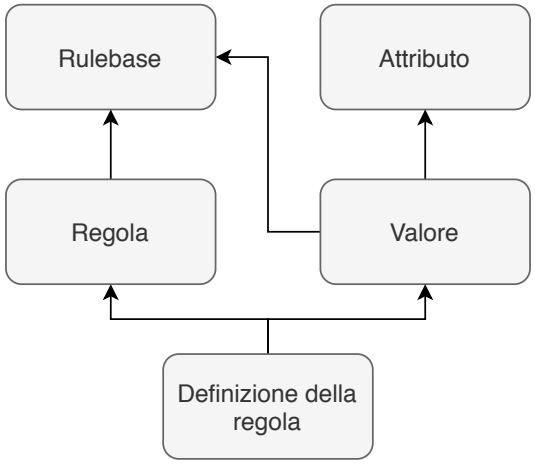


FIGURA 2.2: Struttura del database per il sistema di tabelle decisionali. Per ragioni di segretezza industriale, vengono mostrate solo le entità e le loro relazioni; non vengono invece mostrati i campi di ciascuna tabella.

Le tabelle decisionali del sistema sviluppato sono, come anticipato, *linguistiche*; questa caratteristica è ritenuta essere un vantaggio sia nella fase di acquisizione della conoscenza, che nell'adattività del sistema a contesti differenti.

2.3.2 Mappatura

La componente di mappatura è fondamentale in quanto consente di adattare lo stesso set di regole a contesti differenti. Questo obiettivo viene raggiunto adottando una mappatura delle variabili numeriche in insiemi crisp, identificati da un'etichetta linguistica.

La componente del mapping può essere formalmente analizzata previa definizione di alcuni elementi:

- sia $\mathfrak{D} \subseteq \mathbb{R}$ il dominio della variabile x da mappare;
- siano $A_i,\ i = 1, ..., n$ gli insiemi crisp definiti dal vettore $\boldsymbol{\theta} \in \mathbb{R}^n$.

Allora, la funzione di mapping è:

$$\mu_{\boldsymbol{\theta}} : \mathfrak{D} \to \{0, 1\}, \qquad x \rightarrowtail A_i$$

cioè associa ad ogni $x \in \mathfrak{D}$ il crisp set A_i di appartenenza (con membership pari ad 1).

Nell'implementazione seguita in questo lavoro, i parametri θ_i definiscono l'estremo superiore dell'intervallo crisp; così facendo si ottengono insiemi contigui e non sovrapposti che coprono tutto il dominio della variabile con un numero di parametri contenuto. Un esempio di questa mappatura è riportato nella Tabella 2.1.

Insieme crisp A_i	**Parametro** θ_i	**Intervallo corrispondente**
Molto basso	θ_1	$-\infty < x \leq \theta_1$
Basso	θ_2	$\theta_1 < x \leq \theta_2$
Zero	θ_3	$\theta_2 < x \leq \theta_3$
Alto	θ_4	$\theta_3 < x \leq \theta_4$
Molto alto	θ_5	$\theta_4 < x \leq \theta_5$

TABELLA 2.1: Esempio di mappatura del dominio $\mathfrak{D} = [-\infty, \theta_5]$ con cinque insiemi crisp.

Si noti come questa funzione di mappatura sia, tra tutte quelle possibili, il giusto compromesso tra semplicità, flessibilità ed efficacia gestione dell'incertezza. Risulta evidente che, al variare del vettore $\boldsymbol{\theta}$, varia l'insieme di appartenenza A_i per un certo $\bar{x} \in \mathfrak{D}$.

Grazie alla dipendenza di μ dal vettore $\boldsymbol{\theta}$, la mappatura può avvenire in modalità:

- **manuale**: si effettua assegnando manualmente - sulla base della conoscenza del problema e del contesto - i valori numerici ai parametri θ_i;
- **automatica**: i valori θ_i vengono calcolati tramite algoritmi che analizzano la distribuzione dei dati di input del sistema. Due sono i metodi implementati [30]:
  - tramite *percentili*: il parametro θ_i è assunto essere il $\frac{i}{n} \cdot 100$ percentile della distribuzione della variabile x, con $i = 1, ..., n$;
  - tramite *clustering*: i parametri vengono calcolati tramite l'algoritmo *K-means* [31];
- **semi-automatica**: vengono proposti dei valori di tentativo θ_i (calcolati con i metodi precedentemente descritti), che l'utente può accettare o modificare.

2.3.3 Motore inferenziale

Il motore inferenziale implementato [30] opera in modalità *forward-chaining*. Oltre a fornire i fatti derivati, è suo compito fornire informazioni relative all'interpretabilità del sistema; tali informazioni sono analizzate nel paragrafo 2.3.3.1.

Siccome il sistema di tabelle decisionali consente l'uso della condizione di *don't care*, il motore inferenziale deve occuparsi anche della risoluzione dei conflitti generati dalla possibilità che i termini antecedenti di più regole siano soddisfatti. La scelta adottata in questo caso è quella di applicare, in caso di conflitti, la regola con la specificità maggiore, ovvero quella che ha il minor numero di condizioni *don't care* tra i termini antecedenti.

2.3.3.1 Interpretabilità dei risultati

Il sistema sviluppato consente di capire perché ad un certo fatto sia stato associato un determinato risultato. Questo obiettivo viene raggiunto grazie al modulo di spiegazione, che consente di:

- visualizzare la regola attivata per ciascun fatto noto;
- collegare ad ogni fatto derivato una spiegazione personalizzata, che esplichi in maniera *user-friendly* il ragionamento seguito per arrivare ad una certa conclusione.

Queste caratteristiche riflettono quanto descritto nel paragrafo 1.2.1.2. Questo è utile per due ragioni:

- nella fase di debugging si possono identificare - e correggere - più agevolmente eventuali errori presenti nella *knowledge base*;
- nell'ottica di una continua interazione dell'utente con il sistema, l'interpretabilità dei risultati aumenterà la fiducia che l'utente ripone nel sistema.

2.3.3.2 Inferenza multi livello

Siccome i problemi decisionali da dover risolvere sono arbitrariamente complessi, vi è la necessità di poter effettuare inferenze multiple: questo significa usare i fatti derivati da una tabella decisionale come fatti noti per un'altra. Questa funzionalità viene perseguita grazie all'implementazione dell'architettura *blackboard*, descritta nel paragrafo 1.5.1. Sostanzialmente, il motore inferenziale, durante l'applicazione di una tabella decisionale, legge e scrive i fatti (noti e derivati, rispettivamente) sulla *blackboard*. In tale modo risultano sempre essere disponibili per la consultazione e per l'utilizzo.

2.3.4 Modulo di apprendimento

Il modulo di apprendimento consente al sistema di migliorare le prestazioni - e quindi minimizzare gli errori commessi - nel corso del suo utilizzo grazie all'ac-

quisizione di dati. Per un sistema basato sulla conoscenza come quello sviluppato in questo lavoro, sono state individuate due principali fonti di errore. Queste sono:

- un'errata **rappresentazione della conoscenza**, che può derivare da:
  - logica delle regole sbagliata;
  - mancata identificazione degli attributi caratterizzanti il problema;
- una scorretta **mappatura dei valori**.

La componente di apprendimento implementata consente l'adattamento del set di regole al contesto nel quale esso viene applicato attraverso la modifica dei parametri di mappatura $\boldsymbol{\theta}$, minimizzando dunque la seconda fonte di errore. Questa scelta si deve al fatto che in questo lavoro si può supporre di avere a disposizione la corretta conoscenza[4] del problema in esame. Inoltre, il lavoro in questa tesi si prefigge tra gli altri obiettivi anche quello di investigare le proporzioni tra le fonti di errori descritte; in particolare, si vuole capire quanto la fase di acquisizione della conoscenza e quanto quella di mappatura influiscano sull'accuratezza complessiva del sistema. Ancora, data per certa una *knowledge base*, si vuole stimare di quanto possano migliorare le prestazioni attraverso l'ottimizzazione della mappatura.

La componente di apprendimento è costituita da due moduli, che agiscono sinergicamente al fine di migliorare le prestazioni del sistema decisionale. Di seguito se ne descrivono le caratteristiche.

2.3.4.1 Feedback

Il feedback, ovvero il risultato desiderato per un certo esempio, è un elemento necessario per la fase di apprendimento in quanto i problemi sui quali si prevede di applicare il sistema non consentono di avere a disposizione tali risultati. Il feedback dunque consente di ottenere il valore desiderato attraverso l'interrogazione dell'utente. Tramite questo meccanismo si costruisce progressivamente un dataset di esempi per effettuare la fase di apprendimento supervisionato del modello.

Siccome fornire feedback sulla totalità degli esempi presenti tra i fatti noti non è possibile (a causa del numero arbitrariamente grande di esempi), è necessario definire alcune metodologie per scegliere quali esempi presentare all'utente. L'obiettivo della fase di feedback è quindi carpire il maggior numero di informazioni possibili (e maggiormente utili nella fase di addestramento) con il minor numero di esempi proposti.

Sono state individuate due possibili soluzioni:

[4] Ciò non toglie che possano essere implementate in futuro alcune soluzioni per la costruzione e la modifica della *knowledge base*.

- proporre un certo numero di esempi scelti a caso;
- proporre gli esempi più significativi.

Il concetto di «significatività» viene analizzato di seguito.

Se si suppone che l'incertezza nella previsione sia derivante dalla mappatura delle variabili, è lecito ritenere più significativi gli esempi che abbiano i valori di uno o più attributi in prossimità dei limiti degli insiemi crisp. Il metodo applicato in questo lavoro si basa dunque su questa osservazione. Nel seguito ne viene descritta la logica.

Si consideri un insieme di fatti noti:

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

dove n sono le variabili che caratterizzano il problema ed m sono gli esempi contenuti nei fatti noti; ogni riga della matrice $\boldsymbol{X}$ costituisce quindi un esempio $\boldsymbol{x_i}$. Applicando la tabella decisionale su questi dati, si ottiene un vettore risultati $\boldsymbol{Y}$ costituito da:

$$\boldsymbol{Y} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{Bmatrix}$$

Si applichi ora una quantità stocastica di rumore ad ogni singola variabile di ciascun esempio. La matrice che ne deriva è:

$$\tilde{\boldsymbol{X}} = \begin{bmatrix} \tilde{x}_{11} & \tilde{x}_{12} & \dots & \tilde{x}_{1n} \\ \tilde{x}_{21} & \tilde{x}_{22} & \dots & \tilde{x}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{x}_{m1} & \tilde{x}_{m2} & \dots & \tilde{x}_{mn} \end{bmatrix} = \begin{bmatrix} x_{11} + \xi_{11} & x_{12} + \xi_{12} & \dots & x_{1n} + \xi_{1n} \\ x_{21} + \xi_{21} & x_{22} + \xi_{22} & \dots & x_{2n} + \xi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} + \xi_{m1} & x_{m2} + \xi_{m2} & \dots & x_{mn} + \xi_{mn} \end{bmatrix}$$

Applicando la stessa tabella decisionale su $\tilde{\boldsymbol{X}}$, si ottiene un vettore risultati:

$$\tilde{\boldsymbol{Y}} = \begin{Bmatrix} \tilde{y}_1 \\ \tilde{y}_2 \\ \vdots \\ \tilde{y}_m \end{Bmatrix}$$

Confrontando ora $\boldsymbol{Y}$ ed $\tilde{\boldsymbol{Y}}$, si cercano gli esempi $\boldsymbol{x_i}$, $i = 1, ..., m$ tali che $y_i \neq \tilde{y}_i$; questi esempi sono quelli che, sottoposti ad una perturbazione casuale, hanno

fornito un risultato diverso rispetto al caso non perturbato. Quindi, il feedback su questi elementi risulta, in linea teorica, assumere una maggiore significatività rispetto al feedback sugli altri.

Questa supposizione può essere verificata solamente applicando l'apprendimento con entrambi i meccanismi di feedback e studiandone le prestazioni. Il risultato aspettato è che l'apprendimento effettuato sugli esempi interessanti permetta di raggiungere prestazioni migliori rispetto al learning effettuato su esempi non interessanti, a parità di numero di campioni.

La quantità ξ_{ij} è una perturbazione stocastica attorno allo zero, proporzionale al range dell'attributo per la quale viene calcolata e ad un coefficiente di rumorosità ψ:

$$\xi_{ij} \propto \left[\max_{i=1,\dots,m} (\boldsymbol{x_j}) - \min_{i=1,\dots,m} (\boldsymbol{x_j}) \right] \cdot \psi$$

Il metodo descritto viene identificato nel seguito del lavoro come *robusto*, per richiamare all'attenzione il concetto di introduzione di rumore nel problema.

2.3.4.2 Apprendimento

Il sistema decisionale si può considerare come una funzione di trasferimento h che associa a ciascun esempio $\boldsymbol{x_i}$ un risultato y_i. Si ha quindi:

$$h_{\boldsymbol{\theta}} (\boldsymbol{x_i}) = y_i$$

dove viene messa in evidenza anche la dipendenza del risultato dai parametri $\boldsymbol{\theta} \in \mathbb{R}^n$.

Avendo a disposizione un insieme di k esempi $\boldsymbol{x_j}$ con i relativi feedback $\hat{y}_j$, è possibile definire l'errore commesso dal sistema come funzione dei parametri:

$$e (\boldsymbol{\theta}) = \frac{1}{k} \sum_{j=1,\dots,k}^{y_j \neq \hat{y}_j} 1 \tag{2.1}$$

Si ha dunque:

$$e : \mathbb{R}^n \to [0, 1]$$

La fase di apprendimento si può quindi interpretare come l'ottimizzazione dell'errore $e (\boldsymbol{\theta})$, cioè la ricerca di $\boldsymbol{\theta_{opt}}$ tale da minimizzare la funzione obiettivo.

Sul problema così formulato possono essere fatte alcune osservazioni:

- la definizione della funzione errore sopra data è generale, ovvero può essere applicata a qualunque problema decisionale. Eventualmente possono essere definite altre funzioni errore (che tengono conto della distanza tra le classi) che potrebbero dare risultati migliori, ma perdono in generalità;

- l'ottimizzazione è **mono-obiettivo**: la minimizzazione dell'errore implica la massimizzazione dell'accuratezza del sistema[5];

- data la sua definizione e dato il funzionamento del sistema con logiche crisp, l'errore è una funzione fortemente **discontinua** e **multi-modale**, che presenta molti punti di minimo relativi.

L'ultima osservazione consente di stabilire che si ritiene opportuno l'utilizzo di algoritmi che non si basano sull'utilizzo del gradiente [32]; tra questi, si sceglie di usare il *simplex non lineare* (detto anche *Nelder-Mead*), la cui descrizione è riportata per completezza in Appendice A. Brevemente, le ragioni che hanno portato alla scelta di questo algoritmo sono le seguenti:

- non è basato sul calcolo del gradiente della funzione obiettivo;

- è abbastanza robusto, cioè non si intrappola nei minimi locali;

- ha una buona velocità di convergenza;

- è relativamente semplice da implementare.

La correttezza di queste assunzioni viene verificata nel Capitolo 3, nel quale il metodo descritto viene applicato a dei casi di studio.

2.4 Potenzialità e limiti

La realizzazione del sistema descritto in questo capitolo presenta buone potenzialità, soprattutto in riferimento al contesto rispetto al quale è stato sviluppato. Infatti esso consente di:

- sfruttare la conoscenza espressa sotto forma di regole;

- adattarsi in base al contesto nel quale viene utilizzato;

- fornire spiegazioni sui risultati.

Appaiono tuttavia evidenti alcuni limiti, dovuti alle scelte strutturali, di seguito elencati:

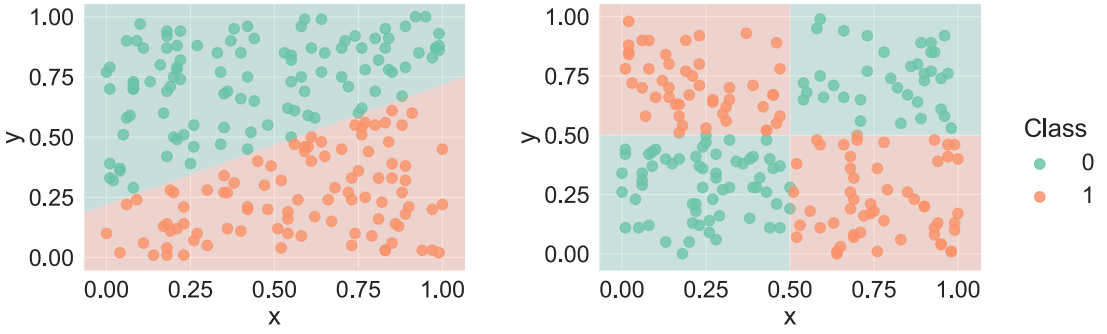
[5]Non si tiene conto della misura di *recall*; ciò potrebbe essere un problema solamente nel caso di problemi con una grande sproporzione tra le classi.

- la classificazione risulta efficiente solo nei problemi che presentano classi separabili con un **numero limitato di iperpiani paralleli agli assi coordinati** nello spazio degli attributi (Figura 2.3b). Siccome ogni regola nella tabella decisionale definisce un iperpiano e visto che il numero di iperpiani può essere al più uguale al numero di esempi (cioè si definisce una regola per ogni esempio), un numero «limitato» di iperpiani significa:

  $$\frac{n_{regole}}{n_{esempi}} \ll 1$$

  Se questa condizione non dovesse essere verificata, si è in presenza di *overfitting*.
  Questo limite è dovuto al fatto che i termini antecedenti delle regole sono congiunzioni logiche di valori per i singoli attributi e non una loro combinazione;

- non vi è una misura della plausibilità della previsione - o della possibilità di appartenenza a ciascuna classe - a causa della scelta di adottare insiemi crisp e non fuzzy. Il sistema appartiene alla categoria dei classificatori *hard* (che si differenziano da quelli *soft*).



(a) (b)

FIGURA 2.3: Dataset linearmente separabile (a) e non linearmente separabile (b) in due dimensioni. Il caso (b) è un esempio di dataset separabile con un numero finito (4) di iperpiani paralleli agli assi coordinati; risulta essere facilmente risolvibile dal sistema sviluppato.

Con riferimento alla Figura 2.3, il sistema risolve facilmente problemi simili a (b): la definizione di 4 regole permette di suddividere il dominio nelle regioni evidenziate e quindi di identificare correttamente le classi. Per risolvere (a) invece sono possibili due metodi:

- trasformare i dati in modo che la linea di separazione sia parallela ad uno degli assi coordinati, così da ricondursi alla tipologia (b);

- approssimare le regioni di piano con un certo numero di rettangoli con lati paralleli agli assi: maggiore il numero di rettangoli, migliore sarà l'accuratezza del sistema. Da evidenziare che un numero maggiore di rettangoli comporterebbe un numero maggiore di regole e quindi una maggiore tendenza all'*overfitting*.

Si ritiene comunque che, in linea con l'obiettivo rispetto al quale il sistema è stato sviluppato, questi limiti non ne precludano il buon funzionamento.

Capitolo 3

Applicazione e risultati

In questo capitolo il sistema sviluppato viene applicato ad alcuni casi d'uso. Dapprima vengono effettuati dei test preliminari per studiare il comportamento delle componenti di feedback e di apprendimento singolarmente, al fine di verificare la validità delle assunzioni del Capitolo 2. Dopodiché i moduli vengono fatti interagire, ricreando l'applicazione prevista del sistema; si cercano - se esistono - delle relazioni tra i meccanismi di feedback e il raggiungimento di prestazioni ottime del sistema. Infine, il sistema viene applicato ad un contesto reale inerente alla *supply chain*, con lo scopo di dimostrarne l'adeguatezza in tale ambiente.

3.1 Test preliminari

Al fine di validare il sistema sviluppato e di verificare le ipotesi formulate nel Capitolo 2, si ritiene necessario ed opportuno condurre alcuni test preliminari. Questi riguardano dei *problemi giocattolo* - molto semplici e senza una vera applicazione pratica - che permettono di verificare l'effettivo funzionamento del sistema e di studiarne il comportamento. In particolare, le componenti che richiedono maggiore attenzione sono il modulo di feedback e il modulo di apprendimento; le sperimentazioni effettuate hanno quindi l'obiettivo di testare singolarmente questi moduli.

3.1.1 Obiettivi

Gli obiettivi che si vogliono perseguire in questa fase preliminare sono i seguenti:

- analizzare il comportamento del modulo di feedback, esaminando i metodi introdotti e investigando il ruolo dei parametri caratteristici;
- verificare la capacità di apprendimento del sistema.

Nel prossimo paragrafo vengono descritti i problemi che sono stati sottoposti al sistema per la sua validazione.

3.1.2 Dataset

I dataset utilizzati per la fase di validazione soddisfano le seguenti caratteristiche:

- riguardano problemi di classificazione;
- hanno un numero limitato di attributi;
- la conoscenza relativa al problema è disponibile e/o deducibile;
- i risultati desiderati sono disponibili.

L'ultimo punto è fondamentale in quanto solamente così è possibile verificare le prestazioni del sistema ed esprimere un giudizio sull'efficienza delle singole componenti. È tale requisito che preclude la possibilità di testare direttamente il sistema in un contesto di *supply chain*, sebbene sia l'ambiente per il quale sia stato sviluppato.

I problemi utilizzati per il test e la validazione del sistema sono:

- problema dell'OR esclusivo (*XOR*);
- classificazione delle specie di fiori *Iris*.

Di seguito se ne descrivono le caratteristiche.

3.1.2.1 *XOR*

Il problema dell'OR esclusivo, chiamato brevemente *XOR*, è un problema di classificazione binaria dove le variabili non sono linearmente separabili. Si basa sulla definizione dell'omonimo operatore logico binario che restituisce vero se e solo se gli ingressi sono diversi fra loro. Questa definizione viene riassunta nella Tabella 3.1.

x	y	**XOR**
0	0	0
0	1	1
1	0	1
1	1	0

TABELLA 3.1: Tabella di verità per l'operatore logico *XOR* [33].

Trasponendo il problema dal dominio discreto a quello continuo, si ottiene il dataset rappresentato in Figura 3.1. I valori $\bar{x}$ e $\bar{y}$, che definiscono la divisione tra le classi, vengono convenzionalmente fissati a 0.5; è comunque possibile assegnare valori differenti a tali parametri, definendo così un altro dataset.

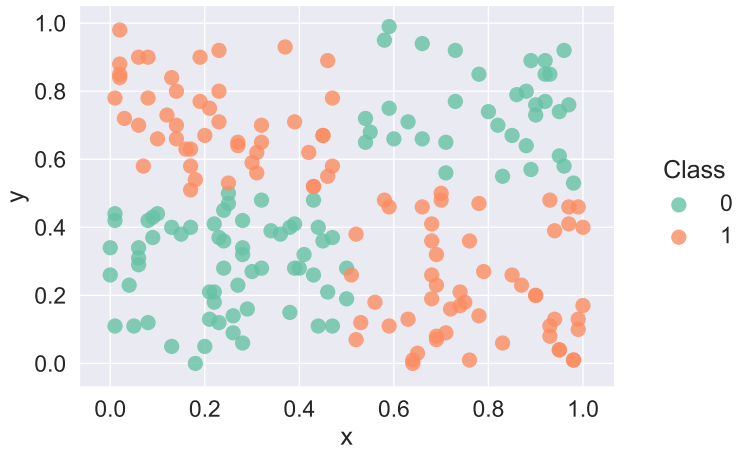


FIGURA 3.1: Dataset per il problema dell'*XOR* con $\bar{x} = \bar{y} = 0.5$.

La fase di acquisizione della conoscenza in questo caso è banale, poiché deriva dalla definizione stessa dell'operatore *XOR*; la tabella decisionale che ne deriva viene riportata in Tabella 3.2. Appare evidente che la mappatura corretta dei valori «poco» e «tanto» è dipendente dai parametri $\bar{x}$ e $\bar{y}$ utilizzati nella definizione del problema.

x	y	**Classe di appartenenza**
Poco	Poco	0
Poco	Tanto	1
Tanto	Poco	1
Tanto	Tanto	0

TABELLA 3.2: Tabella decisionale implementata nel sistema per il dataset *XOR*.

Il numero totale di esempi disponibili in questo dataset è $n = 1000$.

3.1.2.2 *Iris*

Il dataset *Iris* è senz'altro il più utilizzato come test preliminare per i classificatori [34]. Il compito prevede di riconoscere la specie di alcuni fiori in base alle loro

caratteristiche geometriche. Si tratta di una classificazione multi-classe, in quanto sono tre le specie da riconoscere. Queste sono:

- *Iris setosa*;
- *Iris versicolor*;
- *Iris virginica*.

Gli attributi che caratterizzano il problema sono quattro:

- lunghezza e larghezza dei petali;
- lunghezza e larghezza dei sepali.

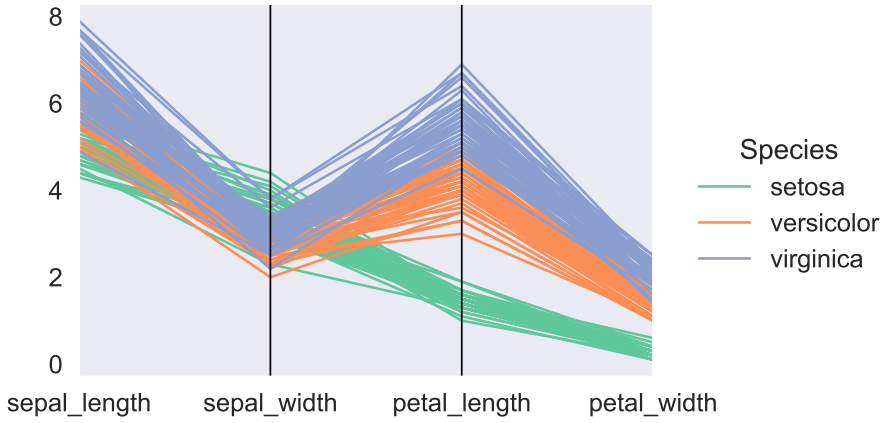


FIGURA 3.2: Visualizzazione del dataset *Iris* in coordinate parallele. Si noti come gli attributi più discriminanti siano la lunghezza e la larghezza del petalo.

La fase di acquisizione della conoscenza viene effettuata grazie all'analisi esplorativa del dataset, della quale la Figura 3.2 ne è un esempio significativo. Il set di regole così ricavato è riportato in Tabella 3.3.

Lunghezza		**Larghezza**		**Specie**
Petalo	Sepalo	Petalo	Sepalo	
Bassa	*	Bassa	*	Setosa
*	*	Bassa	Alta	Versicolor
*	*	Media	*	Versicolor
*	*	Alta	*	Virginica

TABELLA 3.3: Tabella decisionale implementata nel sistema per il dataset *Iris*.

Il numero totale di esempi presenti in questo dataset è $n = 150$.

3.1.3 Feedback

Al fine di validare, almeno in linea di principio, le metodologie di feedback descritte nel paragrafo 2.3.4.1, vengono condotte alcune sperimentazioni. In particolare si studia l'influenza del coefficiente di rumorosità ψ e si definisce una metrica per stabilire la significatività dei feedback proposti. Tutti i test e le osservazioni presentate in questo paragrafo si riferiscono ad analisi effettuate sul solo modulo di feedback; le conclusioni devono quindi essere successivamente verificate osservando l'influenza che l'adozione dei diversi meccanismi ha nel processo di apprendimento.

3.1.3.1 Influenza del coefficiente di rumorosità

Per prima cosa si vuole analizzare il comportamento del metodo robusto al variare del coefficiente di rumorosità ψ. Vengono dunque condotti degli esperimenti sul dataset *XOR*, che consentono di individuare gli esempi proposti come feedback. Di seguito si riportano i risultati di questa analisi.

Nella Figura 3.3 viene evidenziata la sostanziale equivalenza dei meccanismi di feedback casuale e robusto, applicati al dataset *XOR*, quando il coefficiente di rumorosità ψ nel metodo robusto è pari ad 1; in entrambi i casi gli elementi proposti sono distribuiti in modo uniforme in tutto il dominio.

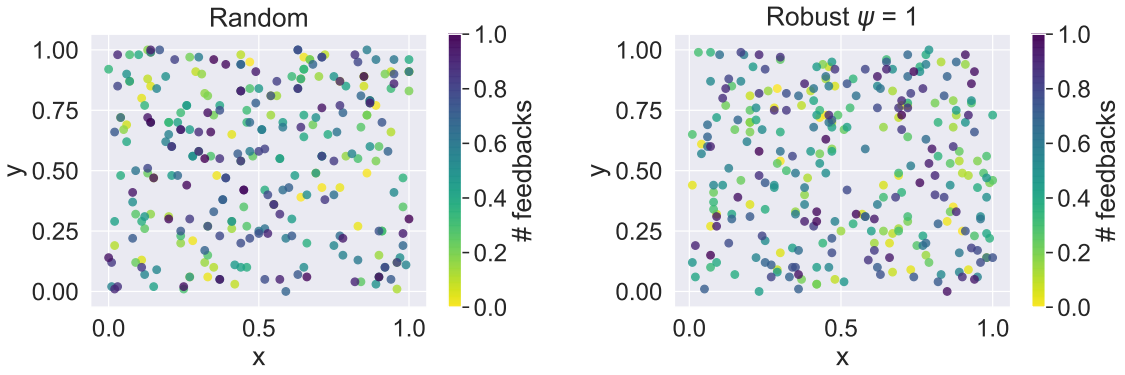


FIGURA 3.3: Elementi proposti come feedback per il meccanismo casuale (a) e robusto con coefficiente di rumorosità $\psi = 1$ (b) applicati al dataset *XOR*. La grandezza # $feedback$ identifica la sequenza temporale con la quale gli esempi vengono proposti, normalizzata rispetto al numero totale degli elementi del dataset; non è presente alcun trend.

Nella Figura 3.4 invece si mostra il comportamento del metodo robusto al variare di ψ; si noti in questo caso come, al diminuire di tale parametro, i feedback

proposti si distribuiscano in prossimità dei limiti degli insiemi degli attributi (in questo caso $\bar{x} = \bar{y} = 0.5$).

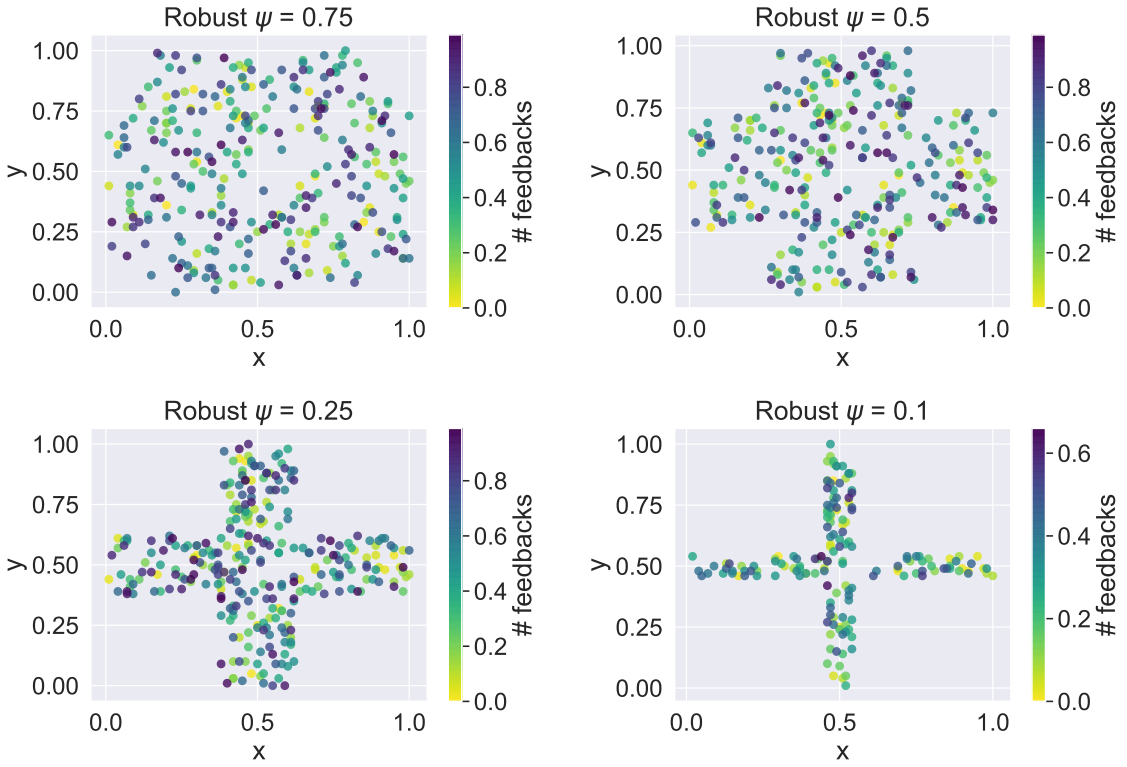


FIGURA 3.4: Feedback proposti nel dataset *XOR* con differenti valori del coefficiente di rumorosità ψ. Si noti in questo caso come gli elementi proposti si distribuiscano in prossimità dei limiti $\bar{x} = \bar{y} = 0.5$ col diminuire di ψ. Non si palesa alcuna dipendenza da # $feedbacks$.

Sebbene non si possa ancora definire quale meccanismo di feedback sia più opportuno usare durante la fase di apprendimento (se ne esiste uno), è certamente possibile formulare alcune osservazioni sul comportamento di questi, analizzando i grafici delle Figure 3.3 e 3.4:

- il metodo **random** non segue alcun criterio particolare nella scelta dell'elemento del quale chiedere il feedback; per questo motivo si ritiene che la sua principale caratteristica sia quella di poter esplorare il dominio. Si ipotizza dunque che possa essere adatto nelle prime fasi di apprendimento (ovvero quando $\boldsymbol{\theta} \neq \boldsymbol{\theta_{opt}}$) in modo da avere una definizione grossolana, ma uniformemente distribuita della funzione errore;

- il metodo **robusto** concentra i feedback negli intorni dei valori θ_i, con la dimensione dell'intorno dipendente dal coefficiente ψ. Si ritiene che que-

sta caratteristica possa essere utile nelle fasi avanzate dell'apprendimento, quando $\boldsymbol{\theta} \simeq \boldsymbol{\theta_{opt}}$, al fine di raffinare la definizione della funzione in un intorno del punto di ottimo.

Euristicamente, nell'ottica dell'approccio sinergico tra meccanismo di feedback ed ottimizzazione, potrebbe essere favorevole un approccio **adattivo**, che abbia un comportamento casuale nelle prime iterazioni dell'apprendimento e che concentri i feedback negli intorni dei parametri θ_i mano a mano che si approccia il punto di ottimo. Una soluzione è quella di far variare il coefficiente di rumorosità ψ in funzione del numero di feedback ricevuti dal sistema. Questo risultato si può ottenere introducendo una dipendenza del coefficiente di rumorosità ψ dal numero di feedback dati. Si fa notare che, così facendo, si introduce anche il concetto di *confidenza* del sistema: maggiore è il numero di feedback dati, maggiore è la confidenza, minore è la quantità di rumore aggiunta e minore è la probabilità di chiedere altri feedback.

La legge di variazione del coefficiente di rumorosità in funzione del numero di feedback che è stata adottata è la seguente:

$$\psi = (1 - \#\, feedbacks)^{\nu}$$

dove $\nu \in \mathbb{R}^+$ prescrive la rapidità della variazione di ψ in funzione del numero di feedback. In Figura 3.5 viene riportato l'andamento di ψ per alcuni valori di ν.

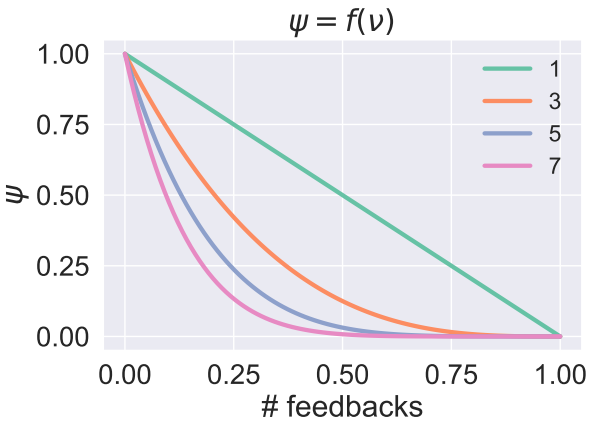


FIGURA 3.5: Coefficiente di rumorosità ψ in funzione del numero di feedback, per diversi valori di ν.

In Figura 3.6 invece viene mostrato il meccanismo di feedback adattivo applicato al problema *XOR*. Il comportamento di questo metodo può essere così descritto:

- con un basso # $feedbacks$ è sostanzialmente analogo al metodo casuale;

- con un alto # *feedbacks* si ha un comportamento simile a quello robusto;
- l'iperparametro ν definisce la rapidità di passaggio tra un comportamento e l'altro.

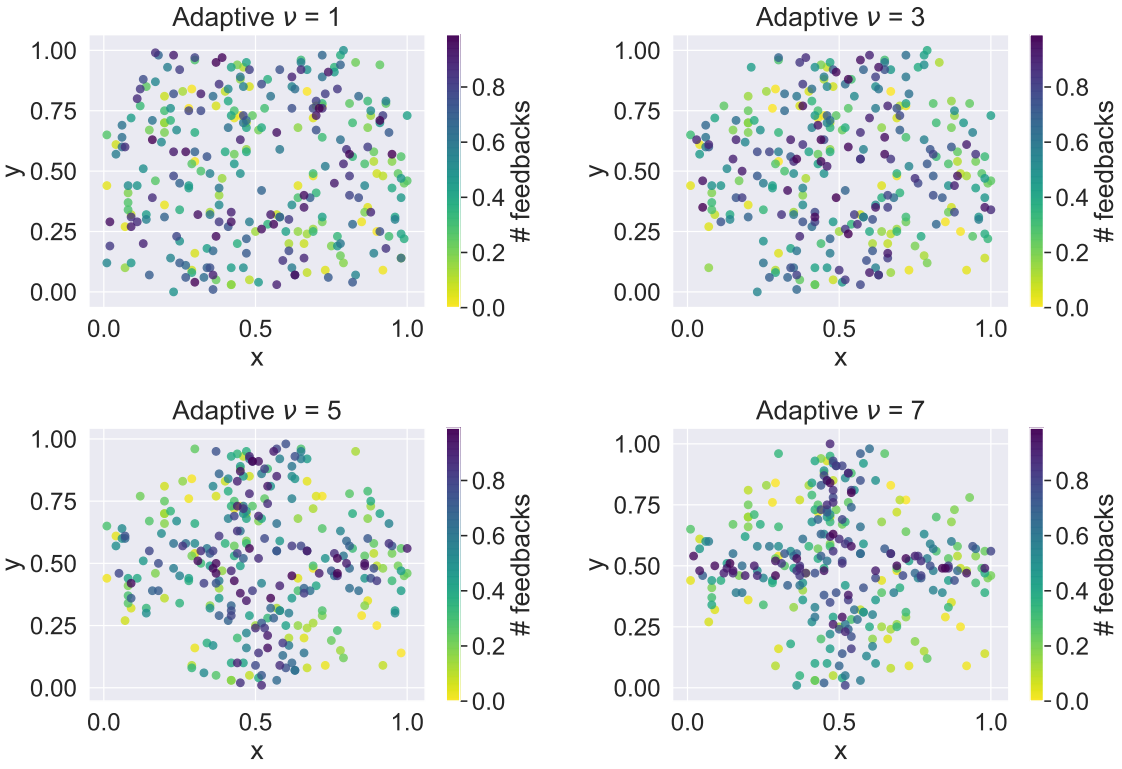


FIGURA 3.6: Metodo di feedback adattivo applicato al dataset *XOR* con diversi valori dell'esponente ν. Si noti come, con l'aumentare del numero di feedback, gli esempi proposti si concentrano maggiormente nella zona di confine tra le classi; questa tendenza è tanto più marcata quanto maggiore è ν.

3.1.3.2 Significatività dei feedback

Si assume come indicatore della significatività del metodo di feedback il seguente rapporto:

$$errorHitRatio = \frac{\#\ feedback\ su\ predizioni\ sbagliate}{\#\ feedbacks}$$

Ovvero, tra tutti i feedback chiesti, quanti riguardano esempi classificati male. Si ritiene interessante questa grandezza nell'ipotesi che le informazioni utili riguardino gli esempi classificati erroneamente dal sistema. Anche in questo caso

l'effettiva significatività del metodo di feedback può essere valutata accuratamente solo durante l'applicazione simultanea all'apprendimento.

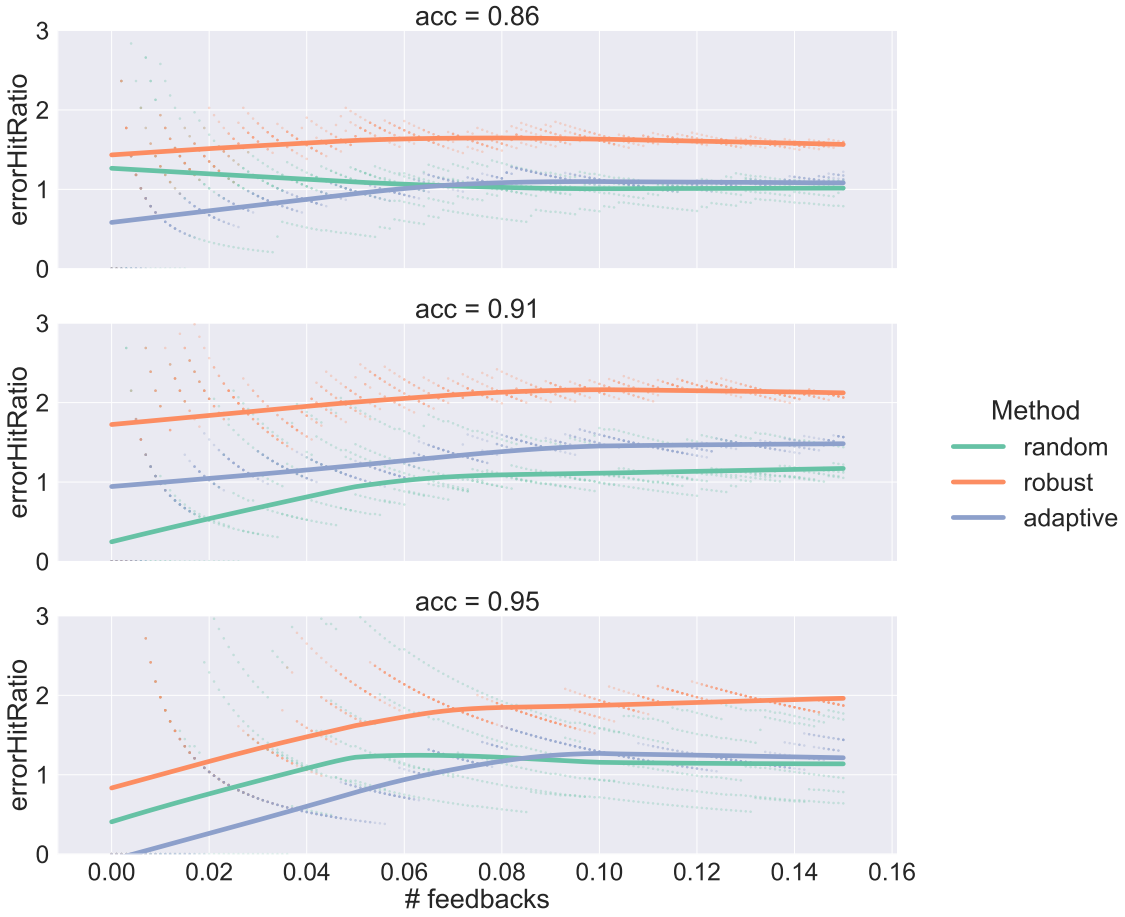


FIGURA 3.7: *ErrorHitRatio* misurato a diverse accuratezze del sistema decisionale applicato al dataset *XOR*. La variabile dipendente è normalizzata rispetto all'errore commesso dal classificatore $(1 - acc)$, mentre quella indipendente rispetto al numero di esempi presenti nel dataset $(n = 1000)$. Le misure sono state effettuate con $\psi = 0.25$ e $\nu = 5$ rispettivamente per il metodo robusto e adattivo. Le linee rappresentano il valore medio riscontrato effettuando 10 simulazioni.

Si nota dalla Figura 3.7 che il metodo random tende al valore 1 più rapidamente del metodo robusto: ciò indica che il meccanismo casuale ha, come ci si aspetterebbe, un'efficacia di identificazione degli esempi classificati erroneamente pari all'errore commesso dal classificatore, con la conseguenza logica che esso risulta poco efficace quando l'accuratezza del sistema è elevata. Si nota, invece, come il metodo robusto ottenga un *errorHitRatio* quasi doppio rispetto a quello del metodo casuale, specialmente nei casi dove l'accuratezza è più elevata. Il me-

todo adattivo in questa analisi evidenzia caratteristiche intermedie a quelle degli altri due; questo fatto non sorprende, in quanto il meccanismo adattivo è stato sviluppato proprio per esibire le caratteristiche di entrambi.

La corrispondenza tra un *errorHitRatio* maggiore ed un'effettiva maggiore efficacia viene discussa nel paragrafo 3.2, nel quale si confrontano i risultati ottenuti attraverso l'apprendimento sui feedback proposti dai tre metodi.

3.1.4 Apprendimento

La validazione della sola componente di apprendimento avviene seguendo la procedura qui descritta:

1. si misurano le prestazioni di partenza ottenute grazie alla mappatura automatica sull'intero dataset;
2. si divide il dataset in *training set* (70% del totale) e *test set* (il restante 30%);
3. si ottimizzano i parametri minimizzando l'errore commesso sul *training set*;
4. si valutano le prestazioni del sistema misurando l'accuratezza che si ottiene sul *test set* utilizzando i parametri ottimi.

Si noti come il flusso descritto sia analogo (fatta eccezione per il primo punto) alla logica dell'apprendimento supervisionato classico.

I problemi che vengono sottoposti al sistema sono:

- dataset *XOR* con $\bar{x} \neq \bar{y} \neq 0.5$: in questo modo la mappatura automatica non risolve il problema con un adeguato livello di accuratezza;
- dataset *Iris* classico.

In Tabella 3.4 sono riportate le prestazioni del sistema sui due problemi in esame prima e dopo la fase di apprendimento; si noti come in entrambi i casi l'accuratezza raggiunta sia aumentata in seguito all'ottimizzazione.

Dataset	**Set**	**Accuratezza**	
		Pre	**Post**
XOR	Training	0.643	1
	Test	0.580	1
Iris	Training	0.924	0.962
	Test	0.911	0.933

TABELLA 3.4: Accuratezza del sistema sul *training* e *test set* ottenuta con la mappatura automatica (*pre*) e dopo l'apprendimento su tutto il *training set* (*post*). Le prestazioni sul *test set* sono migliorate in seguito all'ottimizzazione dei parametri effettuata sul *training set*.

Nel dataset *XOR* l'ottenimento di un'accuratezza pari ad 1 non deve sorprendere in quanto il problema è appositamente - e irrealisticamente - semplice.

È interessante, invece, notare che nel caso del dataset *Iris* l'accuratezza raggiunta dal sistema prima dell'ottimizzazione sia molto buona; questo è un punto di forza del sistema, che essendo basato sulla conoscenza è capace di effettuare delle previsioni - anche accurate - senza aver la necessità di dover analizzare dei dati. Tanto migliore è la fase di acquisizione della conoscenza, tanto migliore è l'accuratezza di partenza.

In Figura 3.8 viene rappresentato il miglioramento delle prestazioni ottenuto in seguito all'ottimizzazione dei parametri; esso è definito come:

$$\Delta_{max} = \frac{\left(1 - acc_{pre}\right) - \left(1 - acc_{post}\right)}{1 - acc_{pre}} = \frac{err_{pre} - err_{post}}{err_{pre}} \tag{3.1}$$

dove con il simbolo Δ_{max} si intende che questo è il massimo incremento raggiungibile, in quanto l'apprendimento è stato effettuato su tutti i dati a disposizione.

È chiaro l'aumento delle prestazioni in seguito alla fase di apprendimento, sia sul set di addestramento che su quello di test. Si noti come il miglioramento delle prestazioni ottenuto sul *test set* di *Iris* sia più contenuto rispetto a quanto si verifica sul *training set*; un'ipotesi che spiegherebbe questo fatto è che il set di addestramento non sia pienamente rappresentativo del problema. Analisi successive permetteranno di verificare questa ipotesi.

FIGURA 3.8: Miglioramento delle prestazioni in seguito alla fase di apprendimento.

I risultati di questa sperimentazione consentono comunque di stabilire che l'ottimizzazione dell'errore attraverso il metodo di Nelder-Mead descritto nel paragrafo 2.3.4.2 permette al sistema di apprendere ed adattarsi.

3.2 Influenza dei feedback sull'apprendimento

In questo paragrafo viene studiata l'influenza del meccanismo e del numero di feedback sulla capacità di apprendimento del sistema. In particolare si vuole ottenere una misura delle massime prestazioni raggiungibili dal sistema in funzione del numero di esempi chiesti all'utente, per ogni tipologia di feedback sviluppata.

I problemi sui quali il sistema viene testato sono quelli utilizzati nel paragrafo 3.1.4, in modo tale da avere un riferimento per la misura dell'accuratezza massima raggiungibile.

3.2.1 Numero minimo di feedback

Viene condotta un'analisi per stabilire se esiste un numero minimo di feedback che consente di ottenere un'accuratezza prossima a quella che si ottiene nel caso di apprendimento sulla totalità degli esempi nel set di addestramento. Il flusso di test è il seguente:

1. si misurano le prestazioni di partenza ottenute grazie alla mappatura automatica sull'intero dataset;

2. si divide il dataset in *training set* (70% del totale) e *test set* (il restante 30%);

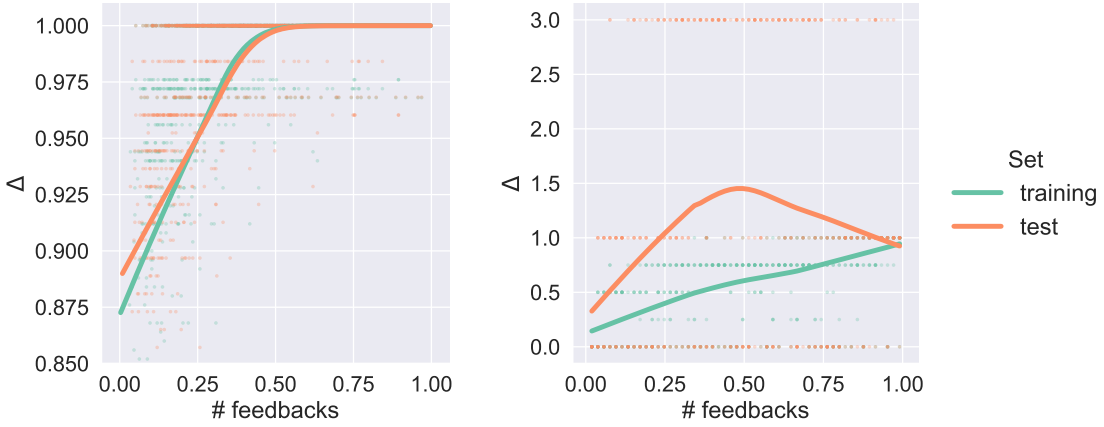
3. si esegue l'apprendimento sul *training set* con un numero crescente di feedback; ad ogni iterazione, l'apprendimento riparte dalla situazione di partenza, in modo tale da studiare solamente l'effetto della dimensione del set di addestramento sulle prestazioni del sistema;

4. si monitora l'errore commesso dal sistema sul *training set* e *test set* all'aumentare del numero di feedback.

L'unica differenza che si ha rispetto al caso di apprendimento supervisionato classico è quindi che la dimensione del *training set* è variabile.

In Figura 3.9 viene riportato l'andamento del miglioramento ottenuto normalizzato rispetto al miglioramento massimo ottenibile Δ_{max} definito in 3.1; la grandezza rappresentata è dunque calcolata come:

$$\Delta\left(\#\, feedbacks\right) = \frac{\left(1 - acc_{pre}\right) - \left(1 - acc_{post}\left(\#\, feedbacks\right)\right)}{\Delta_{max}}$$



(a) (b)

FIGURA 3.9: Miglioramento delle prestazioni normalizzato Δ in funzione del numero di esempi nel *training set*, per i dataset *XOR* (a) e *Iris* (b). Le linee rappresentano il valore medio riscontrato effettuando 10 simulazioni. I feedback vengono scelti con il metodo random.

Innanzitutto si fa notare che Δ assume valori discreti in entrambi i dataset: in Figura 3.9 sono chiaramente visibili le distribuzioni dei punti su ordinate ben definite. Questo è dovuto al fatto che il numero di esempi è finito e, data la definizione 2.1, l'errore assume valori discreti. Tanto maggiore è il numero di esempi nel dataset, tanto più fine sarà la quantizzazione dell'errore.

I risultati sul dataset *XOR* sono in linea con le aspettative: all'aumentare del numero di feedback, l'accuratezza migliora fino a raggiungere il massimo ($\Delta = 1$) con $\#\ feedbacks = 0.5$. Dopo tale valore non si assiste ad alcuna variazione nelle prestazioni. È interessante notare che anche con un numero minimo di feedback (meno di 10) si ha una diminuzione media dell'errore superiore all'85%.

Sul dataset *Iris* si assiste ad un comportamento diverso: innanzitutto l'andamento medio di Δ sul *training set* è pressoché lineare e parte da un valore più basso rispetto al caso precedente. Si ritiene che ciò possa essere spiegato ricordando il valore elevato di accuratezza che si ottiene con la mappatura automatica. Sorprende invece la curva relativa al *test set*: in questo caso si ottengono valori $\Delta > 1$. Ciò indica che l'accuratezza raggiunta sul *test set* ottimizzando l'errore su un sottoinsieme del *training set* risulta essere migliore di quella che si ottiene ottimizzando sul totale. Al fine di capire il perché di questo andamento, è bene precisare alcuni fatti:

- con riferimento alla Tabella 3.4, la mappatura di partenza applicata al *test set* di *Iris* classifica erroneamente:

$$n_{err,pre} = (1 - acc) \cdot 0.3 \cdot n_{tot} = (1 - 0.911) \cdot 0.3 \cdot 150 = 4\ esempi$$

dove $n_{tot} = 150$ sono tutti gli esempi presenti nel dataset e 0.3 è il rapporto tra popolazione del *test set* e il totale;

- la mappatura ottimizzata su tutto il *training set* (Tabella 3.4) permette di classificare erroneamente $n_{err,post} = 3$ *esempi* sul set di test;
- qualsiasi variazione nella classificazione di alcuni elementi del set di test determina dei valori $\Delta \in \mathbb{Z}$, poiché $n_{err,pre} - n_{err,post} = 1$.

Infatti, come si nota dalla Figura 3.9b, le misure relative al miglioramento nel *test set* assumono valori discreti molto sparsi ($\Delta = \{0, 1, 3\}$); la linea media assumerà dunque valori intermedi a questi. Siccome nel corso dell'ottimizzazione sono state incontrate delle situazioni che permettono di classificare erroneamente un solo esempio nel *test set*, il grafico presenta dei punti con $\Delta = 3$. Per questo motivo la curva media assume valori $\Delta > 1$.

Si ritiene che le cause scatenanti di questa situazione atipica siano:

1. la bassa numerosità del dataset *Iris* ($n = 150$), specie se confrontata a quella dell'altro caso di studio ($n = 1000$);
2. una non perfetta omogeneità tra il set di addestramento e di test: ciò implica che ad un certo miglioramento delle prestazioni nel *training set* non corrisponde un analogo aumento dell'accuratezza nel *test set*;
3. l'ottenimento di prestazioni già buone attraverso la mappatura automatica.

La seconda causa, che è una conseguenza della prima, potrebbe essere eliminata effettuando svariate prove mescolando di volta in volta i due set; mediando i risultati l'effetto dovuto alla disomogeneità dei set dovrebbe annullarsi o quantomeno ridursi.

I punti 1 e 3 assieme determinano una quantizzazione grossolana dell'errore - in particolare sul *test set* - che infine causa il comportamento anomalo descritto.

A conferma di queste ipotesi, c'è il fatto che nel dataset *XOR* non si presenta alcun problema di questo tipo, proprio grazie alla maggiore numerosità ed alle prestazioni di partenza più scarse.

Per quanto detto, si ritengono poco significative le indicazioni fornite dalla Figura 3.9b e poco attendibili le osservazioni derivanti dallo studio del dataset *Iris*, in particolare quelle sul set di test.

3.2.2 Apprendimento incrementale

Si vuole verificare ora il funzionamento del sistema in modalità incrementale: ciò significa che, quando vengono forniti nuovi feedback, l'ottimizzazione dei para-

metri parte dalla situazione ottimale raggiunta all'iterazione precedente. Si vuole investigare anche il comportamento del sistema con i diversi meccanismi di feedback sviluppati.

Il flusso di test adottato è il seguente:

1. si misurano le prestazioni di partenza ottenute grazie alla mappatura automatica sull'intero dataset;
2. si divide il dataset in *training set* (70% del totale) e *test set* (il restante 30%);
3. si esegue l'apprendimento sul *training set* con un numero incrementale di feedback, i quali vengono scelti con uno dei meccanismi descritti in precedenza;
4. si monitora l'errore commesso dal sistema sul *training set* all'aumentare del numero di feedback.

Le principali differenze rispetto al caso precedente di apprendimento supervisionato sono:

- il numero di esempi disponibili sui quali effettuare il learning - il *training set* - aumenta di dimensione durante l'ottimizzazione;
- l'apprendimento è di tipo incrementale.

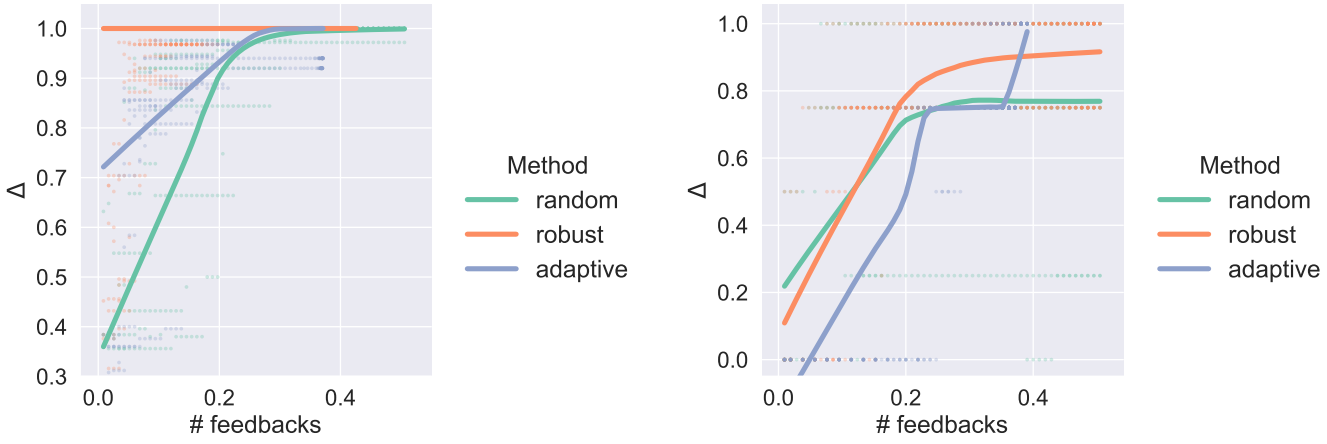


FIGURA 3.10: Miglioramento delle prestazioni normalizzato Δ in funzione del numero di esempi nel *training set* e del meccanismo di feedback adottato per i dataset *XOR* e *Iris*. Le linee rappresentano il valore medio riscontrato effettuando 10 simulazioni. Gli iperparametri adottati sono $\psi = 0.25$ e $\nu = 5$ rispettivamente per i metodi robusto ed adattivo.

Analogamente al caso precedente, anche in Figura 3.10 è presente un'evidente quantizzazione della grandezza Δ, molto più marcata nel caso di *Iris* per le ragioni già discusse.

Osservando il dataset *XOR*, si nota una netta distinzione tra i risultati ottenuti a seconda del tipo di feedback adottato:

- il metodo robusto consente di ottenere mediamente il massimo incremento di accuratezza possibile con un numero minimo di feedback;
- quello casuale fornisce le prestazioni peggiori, in quanto a parità di numero di feedback l'incremento medio delle prestazioni è minore rispetto al caso robusto;
- il meccanismo adattivo presenta caratteristiche intermedie agli altri metodi, a conferma di quanto evidenziato in analisi precedenti.

Questa distinzione non viene riscontrata pienamente nel caso di *Iris*, anche se viene confermata la superiorità del metodo robusto rispetto a quello random. Da sottolineare il comportamento del meccanismo adattivo, che con pochi feedback offre in questa analisi le prestazioni peggiori salvo poi garantire un incremento brusco dell'accuratezza. Questo fatto può essere spiegato se si considera che, con $\#\, feedback = 0.35$ (valore al quale si verifica questo comportamento) corrisponde un $\psi = (1 - 0.35)^5 = 0.12$; ciò significa che gli esempi proposti in quel momento sono estremamente significativi, poiché giacciono in un intorno molto stretto dei θ_i. Si noti inoltre come, successivamente all'impennata, tale metodo non proponga più elementi sui quali richiedere il feedback, segno che il coefficiente ψ è talmente basso che non esistono più esempi «interessanti» nel set. In questo caso si può dire che il sistema ha raggiunto la massima confidenza.

3.2.3 Analisi

L'analisi dei risultati di questo paragrafo permette di effettuare alcune osservazioni sul comportamento generale del sistema:

1. con un numero basso di feedback ($\#\, feedbacks < 0.2$), l'apprendimento incrementale permette di raggiungere, in media, accuratezze più basse rispetto a quello non incrementale (Figura 3.11);
2. con valori superiori di feedback l'apprendimento incrementale offre maggiori margini di miglioramento (Figura 3.11);
3. il metodo robusto si dimostra capace di poter avvicinare le prestazioni ottime anche con un minimo numero di esempi (Figura 3.10).

Le osservazioni 1 e 2 consentono di stabilire che nelle prime iterazioni dell'apprendimento incrementale sarebbe meglio avere a disposizione un maggior numero di feedback. Questo ha senso se si pensa al fatto che, specie nelle prime fasi, una definizione più fitta della funzione errore può agevolare la convergenza verso il minimo globale. Una definizione sparsa dell'errore, invece, può favorire l'intrappolamento nei minimi locali.

Con tali motivazioni si può spiegare anche la terza osservazione: le sperimentazioni effettuate consentono di stabilire che il metodo di feedback robusto ha una maggiore efficacia nell'identificazione degli esempi «interessanti» da proporre, che consentono di definire l'errore in un intorno del minimo globale e quindi di agevolare la convergenza dell'algoritmo di ottimizzazione.

Questi risultati confermano, inoltre, la corrispondenza tra un *errorHitRatio* elevato (Figura 3.7) e una maggiore efficacia del metodo di feedback (Figura 3.10), in quanto il metodo robusto si dimostra essere il migliore in entrambe le analisi.

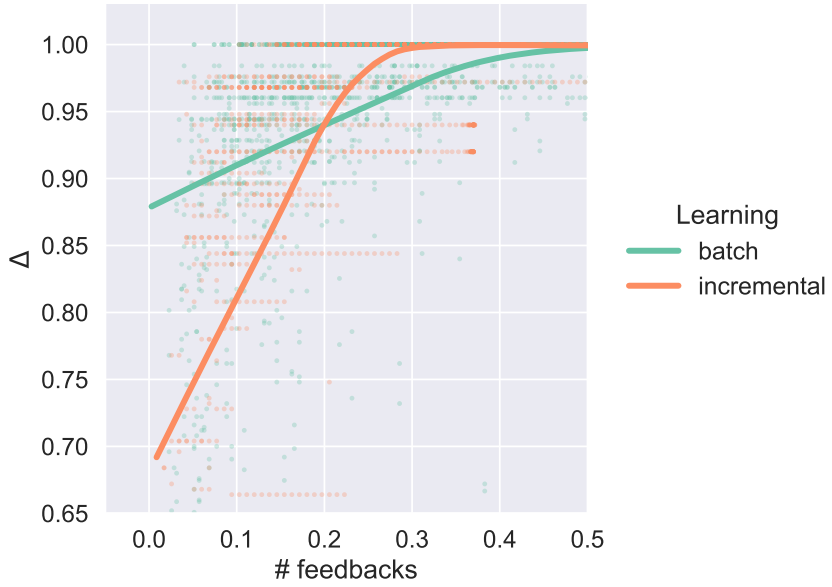


FIGURA 3.11: Confronto tra il miglioramento delle prestazioni normalizzato Δ medio ottenuto con l'apprendimento a lotti e incrementale sul dataset *XOR*.

Per concludere, come già evidenziato precedentemente, il metodo adattivo presenta caratteristiche intermedie agli altri due; probabilmente esso risulterebbe più efficace nel caso in cui $\boldsymbol{\theta}_{opt}$ sia molto diverso da $\boldsymbol{\theta}_{start}$. Se così non è, il comportamento simil-casuale che tale meccanismo presenta inizialmente non solo non è utile, bensì degrada le prestazioni rispetto a quello robusto. È interessante comunque il fatto che riesca ad introdurre il concetto di confidenza del sistema, modulabile attraverso la variazione dell'iperparametro ν.

3.3 Caso d'uso

Avendo verificato la capacità di apprendimento del sistema e avendo caratterizzato i metodi di feedback, è possibile applicare il sistema ad un caso d'uso inerente alla *supply chain*. Per questo lavoro si è scelto di concentrarsi sull'identificazione degli **ordini cliente** (detti anche *commesse* o *corder*), che devono essere monitorati con più attenzione. Questo caso è rilevante nel *business* in quanto:

- il numero di commesse che un responsabile della pianificazione deve analizzare è generalmente molto elevato. Una preselezione delle commesse che riduca il lavoro a carico del responsabile di produzione, rendendolo disponibile per effettuare altre operazioni che richiedono un maggior intelletto, è senz'altro un grosso vantaggio;
- il processo di identificazione è ripetitivo e ricorrente;
- è possibile, in linea teorica, stabilire una logica che sia in grado di risolvere il problema in esame.

Tali caratteristiche corrispondono a quelle descritte nel paragrafo 1.2.1.3, nel quale vengono identificati i problemi convenientemente risolvibili da un sistema esperto.

Questo problema può essere studiato come un problema di classificazione, dove la classe di appartenenza identifica il livello di attenzione da destinare alle commesse.

L'applicazione del sistema decisionale a tale caso d'uso non è tuttavia banale. Infatti:

- la fase di acquisizione della conoscenza, che consente la creazione della tabella decisionale, è problematica;
- non vi è la disponibilità di un dataset reale che consenta di validare la conoscenza acquisita.

I problemi vengono di seguito analizzati nel dettaglio.

3.3.1 Acquisizione della conoscenza

La fase di acquisizione della conoscenza da trasferire al sistema consiste nel ricercare delle regole comunemente adottate dal responsabile della produzione nel processo di identificazione degli ordini cliente da monitorare; questo compito è stato svolto attraverso una serie di interviste a consulenti esperti di dominio.

Sebbene il problema in esame sia relativamente semplice dal punto di vista logico, la fase di acquisizione della conoscenza ha evidenziato alcuni limiti. In particolare:

- non sempre le logiche seguite sono comuni a tutti gli ambienti produttivi. Ciò significa che, tra un contesto e l'altro, non cambia solamente la mappatura degli attributi, bensì la struttura della tabella decisionale stessa; il sistema decisionale sviluppato in questo lavoro non consente, per il momento, di modificare la conoscenza alterando la struttura - e quindi la logica - della tabella di decisione;

- le logiche spesso coinvolgono attributi che non sono esplicitamente presenti nel database; questo si verifica quando le decisioni prese dall'esperto di dominio dipendono da fattori che sono implicitamente contenuti in altri attributi[1];

- le logiche coinvolgono attributi che spesso non sono valorizzati all'interno del database.

Data la presenza di questi limiti, riconducibili al problema della *knowledge acquisition bottleneck* descritto nel paragrafo 1.2.2.1, non è stato possibile ricavare direttamente delle regole generali da inserire nella tabella decisionale; quello che si è ottenuto è piuttosto una serie di attributi discriminanti e la loro importanza relativa. Analizzando queste informazioni e combinandole tra loro è stato possibile definire un set di regole minimale che consente di risolvere con un certo grado di approssimazione il problema in questione; la Tabella 3.5 è dunque il risultato di questo processo. Gli attributi hanno il seguente significato:

- **valore**: è il valore economico dell'ordine cliente;

- **ritardo**: è la differenza tra la data di consegna schedulata e quella pianificata, in giorni. Indica quindi l'entità del ritardo di una commessa rispetto alla situazione pianificata;

- **cliente**: indica se un cliente è importante. Questa informazione è *implicitamente* contenuta nel nome del cliente: la fase di mappatura si occupa di attribuire l'importanza a seconda del nominativo del cliente;

- **stato**: indica se l'ordine cliente è già in produzione (*rilasciato*) o in attesa di essere rilasciato (*pianificato*);

- **inizio**: indica la vicinanza della data di inizio produzione della commessa alla data odierna, in giorni;

- **importanza**: è il termine conseguente della tabella che indica l'importanza da attribuire a ciascun ordine cliente.

[1] Si pensi ad esempio al fatto che alcuni clienti possono essere ritenuti più importanti di altri; questa informazione è implicitamente contenuta nel loro nome.

Valore	Ritardo	Cliente	Stato	Inizio	Importanza
*	*	*	Pianificato	Vicino	Alta
Alto	Basso	Importante	Rilasciato	Vicino	Media
Alto	*	*	Rilasciato	Vicino	Alta
Medio	Alto	Importante	Rilasciato	Vicino	Alta
Medio	Alto	*	Rilasciato	Vicino	Media
Medio	Medio	Importante	Rilasciato	Vicino	Media
Medio	Medio	*	Rilasciato	Vicino	Bassa
Medio	Basso	*	Rilasciato	Vicino	Bassa
Basso	Alto	Importante	Rilasciato	Vicino	Media
Basso	*	*	Rilasciato	Vicino	Bassa
Alto	*	Importante	Pianificato	Lontano	Alta
*	*	*	*	Molto lontano	Bassa
*	*	*	*	Lontano	Media

TABELLA 3.5: Tabella decisionale per l'assegnazione dell'importanza degli ordini cliente.

Questa tabella è una prima approssimazione delle logiche che sottostanno al processo di assegnazione dell'importanza delle commesse: non deve essere quindi vista come un risultato definitivo, bensì come un punto di partenza per analisi e raffinamenti successivi.

Inoltre è stato possibile anche costruire un set di spiegazioni - una per ciascuna regola della tabella decisionale - che esplichino in maniera *user-friendly* il perché di un certo risultato.

3.3.2 Dataset di validazione

La validazione della conoscenza acquisita deve passare attraverso il confronto tra i risultati proposti dal sistema e quelli attesi su un caso reale. Durante la realizzazione del progetto non è però stato possibile reperire un database di produzione che contenesse, oltre agli attributi discriminanti, anche il risultato desiderato. Al fine di verificare comunque le prestazioni del sistema su un caso d'uso inerente alla *supply chain*, si è deciso di costruire un dataset *ad hoc*. Esso ha le seguenti caratteristiche:

- è completo, ovvero contiene tutti gli attributi necessari per l'applicazione della tabella decisionale;
- è realistico, ovvero riflette le caratteristiche di un database di produzione nei valori degli attributi (numerici e non);

- contiene il risultato desiderato, in modo che sia possibile verificare le prestazioni del sistema.

Procedendo in tale modo si combina l'applicazione ad un contesto reale con la possibilità di validare il sistema, misurandone le prestazioni. Il dataset in questione è costituito da $n = 400$ esempi.

3.3.3 Applicazione

Il problema in esame presenta alcune differenze con i casi esaminati fino ad ora, che permettono di stressare maggiormente il sistema e di verificarne l'effettiva funzionalità in contesti produttivi. In particolare, come si evince dalla Tabella 3.6:

- i valori medi tipici degli attributi del problema differiscono di ordini di grandezza (da 10^0 per il ritardo a 10^6 per il valore). Questo potrebbe causare problemi all'algoritmo di ottimizzazione;
- gli attributi sono sia numerici che categorici. Siccome il lavoro presentato in questa tesi permette di ottimizzare solamente attributi numerici, gli eventuali errori dovuti ad una mappatura scorretta di quelli categorici non possono essere diminuiti;
- la dimensione del dominio dell'errore è elevata ($e : \mathbb{R}^9 \to \mathbb{R}$ in questo caso); un'elevata dimensionalità rende più complicata la convergenza dell'algoritmo di Nelder-Mead [35].

Problema	Dimensionalità	Esempi	# regole	Ordine di grandezza
XOR	$\mathbb{R}^4$	1000	4	10^0
Iris	$\mathbb{R}^5$	150	4	10^1
Ordini cliente	$\mathbb{R}^9$	400	13	$10^0 \div 10^6$

TABELLA 3.6: Caratterizzazione dei problemi analizzati.

Risulta quindi ancora più interessante verificare le prestazioni del sistema in tale ambito. Il flusso di validazione, analogo ai precedenti, prevede di:

1. misurare l'accuratezza ottenuta attraverso la mappatura automatica degli attributi sull'intero dataset;
2. valutare l'accuratezza massima raggiungibile effettuando l'apprendimento sull'intero dataset;

3. ripartire dalla situazione data dalla mappatura automatica ed effettuare l'apprendimento incrementale, monitorando il miglioramento delle prestazioni.

In Tabella 3.7 vengono riportate le accuratezze raggiunte con la mappatura automatica e in seguito all'apprendimento sull'intero dataset. Viene definito, in analogia a quanto fatto nelle analisi sugli altri problemi, il massimo incremento ottenibile Δ_{max}. Si noti che anche in questo caso l'ottimizzazione consente un considerevole aumento delle prestazioni.

Accuratezza		Δ_{max}
Pre	**Post**	
0.791	0.941	0.718

TABELLA 3.7: Prestazioni del sistema sul caso d'uso ottenute con la mappatura automatica (*pre*) e con l'apprendimento sull'intero dataset (*post*).

In Figura 3.12 viene mostrato l'incremento normalizzato medio delle prestazioni in funzione del numero e del metodo di feedback. Si noti come i trend riscontrati nell'analisi dei problemi giocattolo descritti nei paragrafi precedenti si verifichino anche in questo caso; in particolare:

- il metodo robusto si conferma, tra tutti quelli sviluppati, il meccanismo che garantisce mediamente il più alto incremento delle prestazioni a parità di numero di feedback;

- maggiore è il numero di feedback dati, maggiore è l'accuratezza raggiunta dal sistema;

- la diminuzione media dell'errore è elevata (intorno al 44% per questo caso) anche con un numero minimo di feedback (meno di 10 elementi).

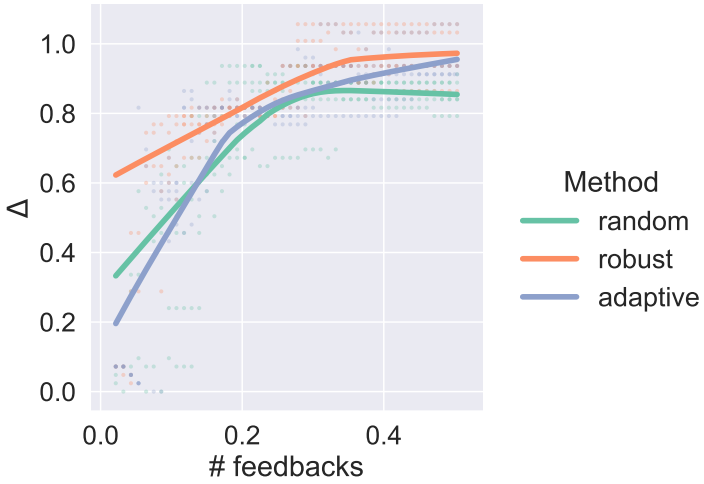


FIGURA 3.12: Miglioramento delle prestazioni normalizzato Δ in funzione del numero di esempi e del meccanismo di feedback adottato per il caso degli *ordini cliente*. Le linee rappresentano il valore medio riscontrato effettuando 10 simulazioni.

Questi risultati permettono di affermare che il sistema decisionale proposto e sviluppato in questa tesi mostra buone prestazioni anche quando applicato a casi inerenti alla *supply chain*. Infatti non sono stati riscontrati limiti nell'applicazione del sistema ad un problema caratterizzato da (Tabella 3.6):

- una dimensione del dominio elevata;
- una logica arbitrariamente complessa, ovvero con un numero maggiore di regole rispetto ai problemi precedentemente analizzati;
- attributi con valori medi che differiscono anche per ordini di grandezza.

Sebbene l'effettiva funzionalità possa essere confermata solamente attraverso l'utilizzo nell'ambiente produttivo, le sperimentazioni condotte in questo paragrafo evidenziano le buone potenzialità del metodo sviluppato a risolvere problemi rilevanti nel *business*, in particolare - ma non limitati a - quelli inerenti alla *supply chain*.

3.4 Risultati

In questo capitolo si è dimostrato che l'utilizzo della conoscenza di dominio sotto forma di regole linguistiche mappate con degli insiemi crisp, unito ad un modulo di apprendimento che minimizza, attraverso la modifica dei suddetti insiemi, l'errore commesso nella classificazione di alcuni elementi «significativi», permette

di ottenere accuratezze elevate nella classificazione di problemi del tipo descritto in 2.4, eliminando la necessità di grandi quantità di dati. A supporto di tale tesi, viene presentato in Tabella 3.8 un riepilogo dei risultati ottenuti, nel quale si evidenziano, per ogni problema analizzato:

- l'accuratezza ottenuta attraverso la mappatura automatica;
- l'accuratezza massima raggiungibile attraverso l'ottimizzazione su tutti gli elementi del dataset;
- il massimo incremento delle prestazioni raggiungibile (Δ_{max});
- l'incremento normalizzato medio ottenuto sottoponendo al sistema 10 feedback scelti con il metodo robusto (Δ_{10});
- l'accuratezza media ottenuta sottoponendo al sistema 10 feedback scelti con il metodo robusto (acc_{10}).

Problema	**Accuratezza**		Δ_{max}	Δ_{10}	acc_{10}
	Pre	**Post**			
XOR	0.643	1	1	1	1
Iris	0.924	0.962	0.500	0.40	0.93
Ordini cliente	0.791	0.941	0.718	0.61	0.87

TABELLA 3.8: Riepilogo delle prestazioni ottenute nei tre problemi analizzati. I valori per i problemi *XOR* e *Iris* si riferiscono al set di addestramento.

Capitolo 4

Conclusioni e sviluppi futuri

L'intelligenza artificiale è destinata ad assumere, nei prossimi decenni, il ruolo di protagonista nel settore industriale [10]. Gli algoritmi di *machine learning* - che stanno divenendo sempre più popolari - non sono in grado di risolvere tutti i problemi industriali; infatti essi:

- necessitano di una grande quantità di dati [20], che non è sempre disponibile;
- forniscono risultati non interpretabili, il che diminuisce la fiducia dell'utente nel modello [17].

I *knowledge-based systems*, d'altro canto, fondano il loro funzionamento sulla base di conoscenza sulla quale sono costruiti [11, 13, 14]; pertanto non hanno bisogno di dati per effettuare delle previsioni. Il grande svantaggio tuttavia è che tale conoscenza è statica e dunque non adattabile a contesti diversi [28]; lo sviluppo di una nuova *knowledge base* per ogni caso non può essere una soluzione fattibile nel mondo industriale [19]. Nasce quindi il bisogno di sfruttare entrambe le fonti di conoscenza [28].

Il sistema decisionale sviluppato in questo lavoro coniuga l'adattabilità derivante delle tecniche di apprendimento automatico al funzionamento basato sulla conoscenza dei sistemi esperti; la conoscenza di dominio viene codificata ed utilizzata per superare il problema di scarsità dei dati. I vantaggi derivanti da questo approccio, dimostrati nel Capitolo 3 di questo elaborato, sono:

- la possibilità di effettuare previsioni - anche accurate - senza la necessità di dover elaborare dei dati, sfruttando la conoscenza di dominio;
- il miglioramento delle prestazioni ottenuto grazie all'utilizzo dei feedback ricevuti, che permettono l'adattamento della conoscenza al caso specifico;

- la capacità del sistema di spiegare i risultati ottenuti, sia attraverso l'introspezione che attraverso la giustificazione.

Si noti come gli ultimi due punti citati siano caratteristiche fondamentali per l'impiego del sistema in un contesto di interazione continua con l'utente [17, 28]; tale applicazione è proprio quella prevista come fine ultimo del progetto, ovvero la realizzazione di un assistente alla produzione all'interno di *CyberPlan*, che aiuti gli utenti nell'identificazione e nella risoluzione delle problematiche riscontrate.

L'unica criticità che si è palesata in questo lavoro è quella relativa alla difficoltà nell'acquisizione della conoscenza di dominio; d'altronde è lecito aspettarsi che al grande vantaggio della non dipendenza del sistema dai dati corrisponda un maggiore impegno su un altro fronte. Questa difficoltà si ritiene possa essere affrontata attraverso uno studio rigoroso dei metodi della *knowledge engineering*; non desta quindi particolare preoccupazione in ottica delle future applicazioni e non pregiudica in alcun modo l'effettiva capacità di funzionamento del sistema decisionale.

4.1 Sviluppi futuri

I risultati presentati nel Capitolo 3 hanno permesso di validare l'architettura del sistema decisionale e gli algoritmi di apprendimento utilizzati. In questo paragrafo vengono presentati possibili sviluppi del sistema che nel corso di questo progetto non hanno avuto l'occasione di essere accuratamente studiati, provati e analizzati. Data la centralità delle componenti di feedback ed apprendimento, le proposte riguardano questi moduli.

Uno sviluppo relativamente semplice, che però richiede un'attenta analisi, è quello di utilizzare una versione di Nelder-Mead adattata per essere più efficiente in spazi ad alta dimensionalità [35]. Questo ha senso poiché, come dimostrato, anche un semplice problema come quello dell'identificazione delle commesse ha un dominio di dimensione elevata; maggiore è la complessità logica del problema, maggiore è il numero di attributi e dunque maggiore è la dimensione del dominio della funzione errore. I benefici attesi sono una maggior velocità di convergenza durante la fase di apprendimento e quindi un minor costo computazionale.

Per quanto riguarda il modulo di feedback, che si è dimostrato essenziale per il buon funzionamento della successiva ottimizzazione, potrebbe essere interessante applicare al sistema un meccanismo di feedback attraverso processi gaussiani ai quali segue un'ottimizzazione Bayesiana [36]. Tale approccio, molto più complesso nell'infrastruttura, permette di costruire un modello surrogato della funzione obiettivo, che viene iterativamente aggiornato e reso più fedele attraverso i feedback. L'ottimizzazione Bayesiana costruisce una distribuzione probabilistica di funzioni - processi gaussiani - che approssima la funzione obiettivo $f(\cdot)$.

Le nuove osservazioni vengono concentrate nelle zone del dominio nelle quali ci si aspetta il più grande incremento di prestazioni e dove l'incertezza è maggiore, sulla base di una funzione di utilità $u(\cdot)$. Il concetto viene rappresentato in Figura 4.1.

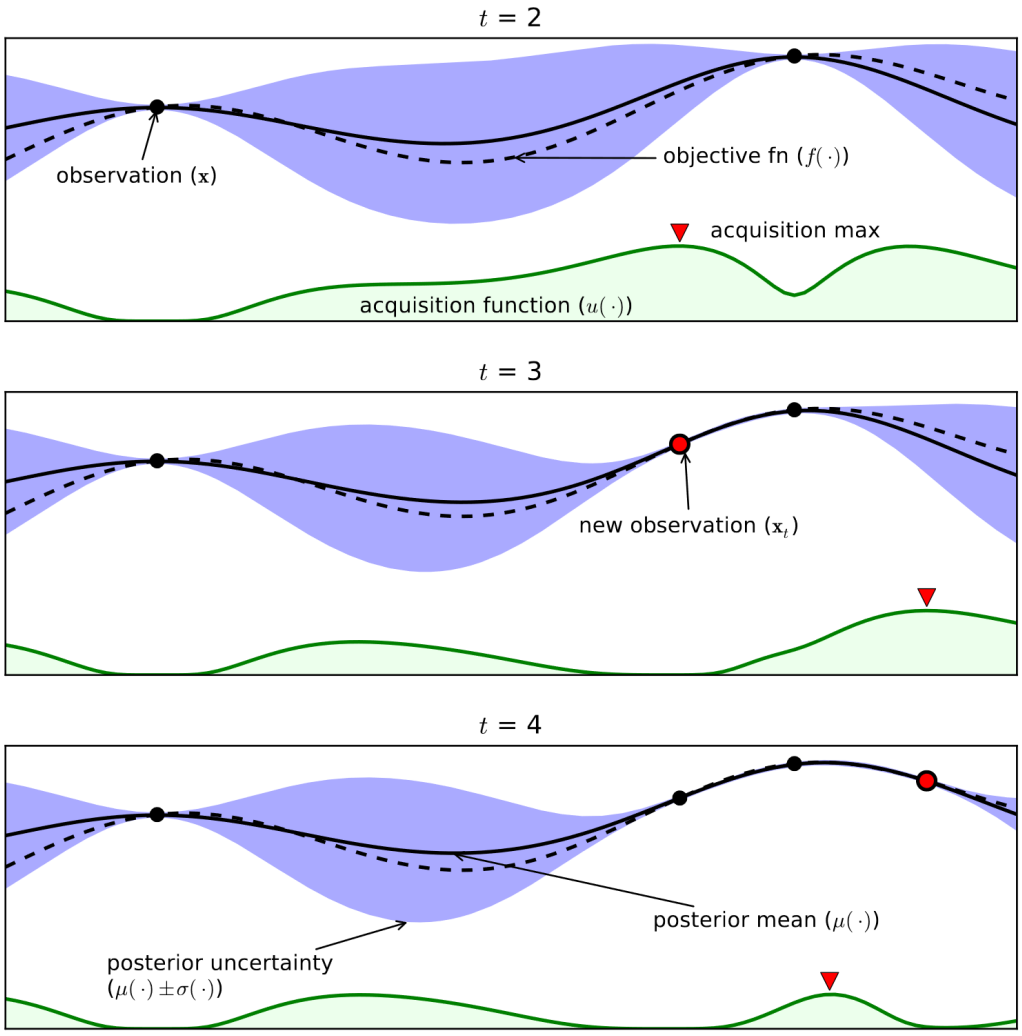


FIGURA 4.1: Ottimizzazione Bayesiana della funzione $f(\cdot)$. Si noti come la nuova osservazione all'iterazione n venga acquisita in corrispondenza del massimo della funzione di acquisizione $u(\cdot)$ all'iterazione $n-1$ [36].

I benefici che ragionevolmente si attendono dall'adozione di questo meccanismo sono:

- una maggiore sinergia tra il modulo di apprendimento e l'ottimizzazione;
- l'eliminazione della componente stocastica nella scelta dei feedback;

- la definizione intrinseca di confidenza del sistema.

4.2 Future applicazioni

Sebbene nel paragrafo 3.3 sia stata dimostrata la funzionalità del sistema in un contesto inerente alla *supply chain*, le sperimentazioni e le applicazioni in tale ambito rappresentano una direzione da esplorare in futuro. In particolare, si identificano due strade:

- applicazione del sistema ad un database di produzione reale;
- risoluzione di altre tipologie di problemi caratteristici nell'ambiente produttivo.

In quest'ultimo caso si ritiene che la fase critica sia legata alla fase di acquisizione della conoscenza e non alla risoluzione del problema in sé. Eventualmente sono da sperimentare, oltre alle interviste dirette, altri metodi per il trasferimento della conoscenza da esperti di dominio al sistema.

Bibliografia

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354—-, 2017.

[2] Q. Huang, "Application of Artificial Intelligence in Mechanical Engineering," vol. 74, no. Iccia, pp. 855–860, 2017.

[3] C. Costa and I. Brandao, "Vibration Analysis of Rotary Machines Using Machine Learning Techniques," *European Journal of Engineering Research and Science*, vol. 4, 2019.

[4] R. Iuganson and M. Vihtonen, "Artificial Intelligence in 3D Printing Real-time 3D printing control," p. 42, 2018.

[5] S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine Learning for Fluid Mechanics," *Annual Review of Fluid Mechanics*, vol. 52, no. 1, pp. 1–32, 2020.

[6] "modeFRONTIER," https://www.esteco.com/modefrontier (Consultato il 19/01/2020), 2018.

[7] L. Tremolada, "Come sarà l'Europa nel 2081? La piramide della popolazione," https://www.infodata.ilsole24ore.com/2017/08/24/sara-leuropa-nel-2081-la-piramide-della-popolazione/ (Consultato il 09/11/2019).

[8] "Population Pyramids of the World from 1950 to 2100," https://www.populationpyramid.net/europe/2020/ (Consultato il 09/11/2019).

[9] G. Corbellini, "Il declino dell'intelligenza," https://www.ilsole24ore.com/art/il-declino-dell-intelligenza–AEJlnLNG (Consultato il 09/11/2019).

[10] J. Bughin, J. Seong, J. Manyika, L. Hämäläinen, E. Windhagen, and E. Hazan, "Tackling Europe's gap in digital and AI," https://www.mckinsey.com/featured-insights/artificial-intelligence/tackling-europes-gap-in-digital-and-ai (Consultato il 09/11/2019).

[11] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., ser. Series in Artificial Intelligence. Upper Saddle River, NJ: Prentice Hall, 2010.

[12] "Turing test," https://en.wikipedia.org/wiki/Turing_test (Consultato il 02/01/2020).

[13] A. A. Hopgood, *Intelligent Systems for Engineers and Scientists (3rd Ed.)*. USA: CRC Press, Inc., 2012.

[14] K. Simon and C. Malcolm, *An Introduction to Knowledge Engineering*. Berlin, Heidelberg: Springer-Verlag, 2006.

[15] J. McCarthy, *Logic Gem*, New Shoes Software Company, Ed. Sterling Castle, 1988.

[16] "Drools," https://www.drools.org/ (Consultato il 09/11/2019).

[17] O. Biran and C. Cotton, "Explanation and Justification in Machine Learning : A Survey," in *IJCAI-17 workshop on explainable AI*, 2017.

[18] J. K. Debenham, *Knowledge Systems Design*. USA: Prentice-Hall, Inc., 1989.

[19] C. Wagner, "Breaking the knowledge acquisition bottleneck through conversational knowledge management," *Information Resources Management Journal*, vol. 19, no. 1, pp. 70–83, 2006.

[20] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated, 2019.

[21] K. Valášková, T. Kliestik, and M. Mišanková, "The Role of Fuzzy Logic in Decision Making Process," in *2nd International Conference on Management Innovation and Business Innovation (ICMIBI 2014)*, 2014.

[22] T. J. Ross, *Logic and Fuzzy Systems*. John Wiley & Sons, Ltd, 2010, ch. 5, pp. 117–173.

[23] L. Erman and V. Lesser, "A Multi-Level Organization For Problem Solving Using Many, Diverse, Cooperating Sources Of Knowledge." in *IJCAI*, 1975, pp. 483–490.

[24] G. Brzykcy, J. Martinek, A. Meissner, and P. Skrzypczyński, "Multi-agent blackboard architecture for a mobile robot," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 4, 2001.

[25] E. Akdeniz and M. Bagriyanik, "A knowledge based decision support algorithm for power transmission system vulnerability impact reduction," *INTERNATIONAL JOURNAL OF ELECTRICAL POWER & ENERGY SYSTEMS*, vol. 78, pp. 436–444, 2016.

[26] N. Walia, H. Singh, and A. Sharma, "ANFIS: Adaptive Neuro-Fuzzy Inference System- A Survey," *International Journal of Computer Applications*, vol. 123, no. 13, pp. 32–38, 2015.

[27] D. W. Aha, "Integrating machine learning with knowledge-based systems," in *Proceedings 1993 The First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, nov 1993, pp. 150–151.

[28] W. Tang, K. Z. Mao, L. O. Mak, and G. W. Ng, "Adaptive Fuzzy Rule-Based Classification System Integrating Both Expert Knowledge and Data," in *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, vol. 1, nov 2012, pp. 814–821.

[29] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter Notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87–90.

[30] D. Korais, "An active learning rule based decision support system for production planning," 2020.

[31] "K-means_clustering," https://en.wikipedia.org/wiki/K-means_clustering (Consultato il 21/01/2020).

[32] A. D. Belegundu and T. R. Chandrupatla, "Optimization Concepts and Applications in Engineering," *Optimization Concepts and Applications in Engineering*, 1999.

[33] "Exclusive or," https://en.wikipedia.org/wiki/Exclusive_or (Consultato il 09/02/2020).

[34] D. Dua and C. Graff, "{UCI} Machine Learning Repository," http://archive.ics.uci.edu/ml (Consultato il 20/11/2019), 2017.

[35] F. Gao and L. Han, "Implementing the Nelder-Mead simplex algorithm with adaptive parameters," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 259–277, 2010.

[36] "Shallow understanding on bayesian optimization," https://towardsdatascience.com/shallow-understanding-on-bayesian-optimization-324b6c1f7083 (Consultato il 19/02/2020).

[37] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[38] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51–56.

[39] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, b. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, 2020.

[40] M. Waskom, O. Botvinnik, J. Ostblom, S. Lukauskas, P. Hobson, MaozGelbart, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, C. Swain, A. Miles, T. Brunner, D. O'Kane, T. Yarkoni, M. L. Williams, and C. Evans, "mwaskom/seaborn: v0.10.0 (January 2020)," https://doi.org/10.5281/zenodo.3629446#.XklbpT0BIxA.mendeley, jan 2020.

[41] APICS The Association for Operations Management, *APICS Operations Manamgent Body Of Knowledge Framework*, 2009.

[42] E. Mameli, "Impiego del software APS CyberPlan per lo sviluppo di un applicativo didattico di Supply Chain Management," 2018.

[43] X. Huang, "Robust simplex algorithm for online optimization," *Physical Review Accelerators and Beams*, vol. 21, no. 10, 2018.

[44] M. Gufar and U. Qamar, "A rule based expert system for syncope prediction," in *2015 SAI Intelligent Systems Conference (IntelliSys)*, nov 2015, pp. 559–564.

Appendice A

Metodo di Nelder-Mead

Il metodo di Nelder-Mead fa parte della categoria degli algoritmi di ottimizzazione che non si basano sull'uso del gradiente. Le caratteristiche di questo metodo sono:

- una maggior robustezza, poiché viene esplorato tutto il dominio della funzione;
- la competitività del metodo per ottimizzazioni con un numero limitato di variabili;
- una maggior flessibilità, perché è possibile ottimizzare funzioni che dipendono da variabili discrete;
- una velocità di convergenza relativamente elevata.

Il nome dell'algoritmo simplex si deve al *simplesso*, ovvero una figura costituita da $n+1$ vertici nello spazio $\mathbb{R}^n$, con n numero di variabili da ottimizzare. Esempi di simplesso sono dunque i triangoli in $\mathbb{R}^2$ ed i tetraedri in $\mathbb{R}^3$.

A.1 Definizione del problema

Al fine di descrivere il funzionamento dell'algoritmo, si ritiene necessario definire il problema di ottimizzazione. Sia:

$$f : \mathbb{R}^n \to \mathbb{R}$$

la funzione da minimizzare, ovvero si vuole trovare:

$$min_{\boldsymbol{x}} \left(f \left(\boldsymbol{x} \right) \right)$$

A.2 Funzionamento

Il metodo di Nelder-Mead segue una semplice ed elegante idea [43]: esso prevede di iniziare con $n+1$ configurazioni (i vertici del simplesso) e migliorare tra queste ad ogni iterazione la soluzione peggiore. Di seguito viene riportata la descrizione dell'algoritmo [43]:

1. inizializzazione del simplesso: le $n+1$ configurazioni possono essere derivanti da un'analisi preliminare (ad esempio un'esplorazione del dominio) o possono essere calcolate partendo da una soluzione perturbandola di una certa quantità su ciascuno degli n assi;
2. calcolo della funzione obiettivo f in ciascuna configurazione;
3. ordinamento dei $n+1$ valori della funzione in ordine crescente (dal migliore al peggiore): $f_1 < ... < f_n < f_{n+1}$ calcolate rispettivamente in $\boldsymbol{x_1}, ..., \boldsymbol{x_n}, \boldsymbol{x_{n+1}}$;
4. definizione del punto medio della faccia del simplesso opposta al vertice $\boldsymbol{x_{n+1}}$: $\boldsymbol{x_c} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x_i}$;
5. calcolo dei possibili candidati: vengono valutati nuove configurazioni sulla linea che congiunge $\boldsymbol{x_{n+1}}$ e $\boldsymbol{x_c}$ seguendo diverse logiche:
   - riflessione: $\boldsymbol{x_r} = \boldsymbol{x_c} + \alpha (\boldsymbol{x_c} - \boldsymbol{x_{n+1}})$;
   - espansione: $\boldsymbol{x_e} = \boldsymbol{x_c} + \beta (\boldsymbol{x_c} - \boldsymbol{x_{n+1}})$;
   - contrazione interna: $\boldsymbol{x_{ic}} = \boldsymbol{x_c} - \gamma (\boldsymbol{x_c} - \boldsymbol{x_{n+1}})$;
   - contrazione esterna: $\boldsymbol{x_{oc}} = \boldsymbol{x_c} + \delta (\boldsymbol{x_c} - \boldsymbol{x_{n+1}})$;
6. valutazione nel punto riflesso $f_r = f(\boldsymbol{x_r})$. Se $f_1 < f_r < f_{n+1}$, la configurazione $\boldsymbol{x_{n+1}}$ viene sostituita da $\boldsymbol{x_r}$ e l'iterazione termina;
7. se $f_r < f_1$, si valuta $f_e = f(\boldsymbol{x_e})$ e si sostituisce $\boldsymbol{x_{n+1}}$ con $\boldsymbol{x_e}$ o $\boldsymbol{x_r}$, a seconda di chi ha il valore minore della funzione. Si termina l'iterazione;
8. se $f_n < f_r < f_{n+1}$, si valuta $f_{oc} = f(\boldsymbol{x_{oc}})$. Se $f_{oc} < f_r$, si sostituisce $\boldsymbol{x_{n+1}}$ con $\boldsymbol{x_{oc}}$ e si termina l'iterazione;
9. se $f_r > f_{n+1}$, si valuta $f_{ic} = f(\boldsymbol{x_{ic}})$. Se $f_{ic} < f_r$, si sostituisce $\boldsymbol{x_{n+1}}$ con $\boldsymbol{x_{ic}}$ e si termina l'iterazione;
10. iterare dal punto 2 fino a quando il criterio di convergenza non viene soddisfatto.

Possibili criteri di convergenza per l'algoritmo sono:

- raggiungimento del massimo numero di iterazioni;
- raggiungimento di una certa tolleranza nel calcolo della differenza tra valore migliore e peggiore nelle $n + 1$ configurazioni.

A.3 Iperparametri

Nella descrizione del metodo sono stati introdotti i parametri $\alpha, \beta, \gamma, \delta$ che condizionano il funzionamento dell'algoritmo. Valori tipici per queste grandezze sono [43, 35]:

$$\begin{gathered}\alpha = 1 \\ \beta = 2 \\ \gamma = 0.5 \\ \delta = 0.5\end{gathered}$$

È comunque importante specificare che l'efficacia dell'algoritmo dipende da tali parametri e dal problema specifico al quale il metodo viene applicato.

Appendice B

Il software APS CyberPlan

In questo capitolo viene descritto il software *CyberPlan*, le sue caratteristiche principali ed i benefici attesi dalla sua adozione. In questo lavoro di tesi *CyberPlan* è stato utilizzato come piattaforma di sviluppo e non come strumento di analisi, nell'ottica futura di integrazione dello strumento sviluppato all'interno del software.

B.1 Descrizione

Il software *CyberPlan* è la soluzione APS[1] progettata da Cybertec che si pone come obiettivo quello dell'ottimizzazione della *supply chain*, migliorando il livello di servizio e riducendo i costi di produzione. La sua naturale applicazione è il settore dell'industria manifatturiera; a seconda degli scenari produttivi, *CyberPlan* offre diverse possibilità:

- nell'ambiente *Make-To-Stock* (MTS) permette di effettuare previsioni sulla domanda e quindi di allineare la produzione su determinati livelli [42];
- nell'ambiente *Assembly-To-Order* (ATO) consente di bilanciare il mix sulla linea di assemblaggio finale [42];
- nell'ambiente *Make-To-Order* (MTO) consente di programmare il carico sulle singole risorse, ottimizzando i costi legati al processo produttivo [42];
- nell'ambiente *Engineering-To-Order* (ETO) permette di garantire i tempi di consegna degli ordini clienti, programmando la produzione in base ai requisiti di ciascuna commessa [42].

[1] Con APS si identificano i sistemi *Advanced Planning and Scheduling*, cioè dei software che utilizzano algoritmi e logiche matematiche per l'ottimizzazione e simulazione della produzione, al fine di risolvere problemi di schedulazione [41].

B.1.1 Caratteristiche

Le caratteristiche principali di *CyberPlan* sono riportate di seguito:

- il database caricato nella memoria RAM garantisce un'elevata velocità, permettendo una continua interazione tra uomo e macchina;
- l'interfaccia grafica potente permette di evidenziare le criticità del piano di produzione, rendendo lo strumento efficace ed al tempo stesso user-friendly;
- l'integrazione con i sistemi ERP consente di essere inserito nel flusso informativo in maniera lineare;
- l'alta configurabilità permette a *CyberPlan* di modellare accuratamente e facilmente le singole realtà produttive;
- la disponibilità in *cloud* e su piattaforma web permette l'accessibilità ai dati da dispositivi diversi.

B.1.2 Benefici

I benefici attesi derivanti dall'adozione del software sono:

- una maggiore visibilità sulla produzione;
- il raggiungimento di un elevato livello di servizio;
- l'innalzamento dei livelli di saturazione e di efficienza;
- la riduzione dei tempi di attraversamento;
- la riduzione del *work-in-process* (WIP) e dell'inventario, con conseguente riduzione dei costi;
- la possibilità di effettuare analisi *what-if*.

Ringraziamenti

Sarei certamente una persona diversa se non fossi circondato da belle persone. E la strada che ho percorso per raggiungere questo traguardo sarebbe stata molto più lunga e impervia se non ci foste stati voi a supportarmi. A tutti voi esprimo la mia gratitudine, perché ognuno di voi ha contribuito con un semplice gesto, una banale chiacchierata, dei consigli, degli spunti di riflessione o dei momenti di sfogo a rendere questo lavoro - e me - migliore. In voi ho trovato sostegno, fiducia e ispirazione.

Un grazie innanzitutto a Helmut e Susanna, per avermi accolto in Cybertec e per avermi dato l'opportunità di confrontarmi con un progetto così interessante ed ambizioso. Un sentito ringraziamento anche al mio relatore, il professor Pozzetto, per essere stato sempre disponibile e cortese. Un grazie anche all'Università degli Studi di Trieste che ha reso possibile l'incontro con Cybertec, grazie al progetto *Talent Acquisition*.

A mamma e papà, per aver supportato moralmente ed economicamente ogni mia decisione, per avermi cresciuto, per avermi instillato il valore dell'ambizione, ma soprattutto per volermi bene. Se sono arrivato fin qui, lo devo a voi.

A mio fratello, per essere la mia figura di riferimento, soprattutto in ambito accademico. Sei uno stimolo costante che mi spinge a mettermi in gioco, a fare di più e farlo meglio.

Ai miei grandi amici di sempre, che siete pronti ad accogliermi con un sorriso anche dopo mesi di mia assenza (ingiustificata, aggiungo).

Last but not least. A Susi. Perché hai alleggerito il peso di questo periodo facendomi scoprire Scrubs. Perché nonostante quello che studio «sia noioso», mi stai ad ascoltare. Perché la tua forza di volontà incentiva la mia determinazione. Perché il tuo coraggio è contagioso. Perché nonostante la distanza che ci separa, ti sento vicina.