ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea in Ingegneria e Scienze Informatiche

Analisi comparativa di metodi per induzione di alberi decisionali

Tesi di laurea in:
Algoritmi e strutture dati

Relatore
Prof. Vittorio Maniezzo

Candidato
Ezmiron Deniku

III Sessione di Laurea
Anno Accademico 2022-2023

Keywords

Machine learning,
alberi decisionali,
classificazione.

Sommario

Nell'ambito del machine learning, gli alberi decisionali (DTs, Decision Trees) rappresentano un importante strumento di apprendimento supervisionato per la classificazione e la regressione. Questi modelli, caratterizzati da una struttura ad albero, offrono un'elevata interpretabilità e sono ampiamente usati per la loro capacità di fornire predizioni accurate e comprensibili.
Questo studio si concentra sull'analisi comparativa di tre algoritmi di induzione degli alberi decisionali: Classification and Regression Trees (CART), Optimal Classification Trees (OCT), e MurTree (MT). CART, un algoritmo euristico, ha dimostrato un'ottima efficienza, ma, anche, una tendenza al sovraadattamento. Al contrario, OCT e MT, algoritmi ottimali, si distinguono per la capacità di generare alberi decisionali con maggiore accuratezza e minore complessità.
Attraverso il lavoro svolto, che ha richiesto fra l'altro l'estensione del codice sorgete di repository pubblici e l'interazione con aziende che commercializzano alcuni dei codici, emerge che OCT e MT tendono a generare alberi con una maggiore precisione rispetto a CART. Inoltre, OCT si distingue per la sua capacità di ridurre la complessità degli alberi, mentre MT si focalizza sull'ottimizzazione globale.
Questo studio fornisce una panoramica dettagliata delle prestazioni e della complessità dei DTs elaborati da questi algoritmi, dimostrando che la scelta del metodo di induzione impatta significativamente sulle prestazioni e sull'interpretabilità degli alberi decisionali. Le conclusioni traggono spunti importanti per la selezione degli algoritmi in base alle esigenze specifiche dell'applicazione.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Capitolo 1

Introduzione

Nella disciplina dell'Informatica, gli alberi decisionali (DTs, Decision Trees) rappresentano un fondamentale modello di apprendimento supervisionato non parametrico, ampiamente impiegato per compiti di classificazione e regressione. Tali strutture decisionali sono noti per la loro facilità di interpretazione da parte degli esseri umani, costituendo uno strumento cruciale nel campo del machine learning per la loro capacità di fornire predizioni accurate e al contempo comprensibili. Questi modelli sono caratterizzati da una struttura ad albero gerarchica, composta da un nodo radice, rami, nodi interni e nodi foglia.

Il presente elaborato si propone di condurre un'analisi dettagliata e un confronto diretto tra tre algoritmi per l'induzione di alberi decisionali, rispettivamente denominati Classification and Regression Trees (CART), Optimal Classification Trees (OCT) e MurTree (MT).

CART rappresenta uno dei pilastri fondamentali nell'ambito dell'elaborazione di alberi decisionali. Questo algoritmo, proposto da Breiman et al. nel 1984 [1], adotta una strategia top-down per la determinazione delle partizioni. Questa suddivisione è guidata da criteri che mirano a massimizzare l'omogeneità all'interno dei gruppi ottenuti, risultando in una soluzione localmente ottima, il che rende

questo algoritmo di tipo euristico [2]. Va notato che, sebbene fosse stato dimostrato già nel 1976 che la costruzione di tali alberi è un problema NP-hard [3], al tempo era computazionalmente impraticabile ottenere una soluzione ottimale.

Successivamente con lo sviluppo di tecniche di ottimizzazione miste intere (MIO), si è avvertita la necessità di realizzare l'algoritmo Optimal Classification Trees (OCT), presentato da Bertsimas e Dunn nel 2017 [4]. A differenza di CART, OCT si propone di elaborare l'intero albero decisionale, mirando a una soluzione ottimale globale. Quando questo avviene si nota che una rappresentazione più accurata dei dati porta a una migliore generalizzazione su dati non osservati [5][6].

Gli alberi decisionali ottimali a livello globale rivestono particolare importanza in contesti socialmente sensibili, poiché questa soluzione possiede un ruolo cruciale nel garantire l'equità [7]. Inoltre, in alcune applicazioni, la dimensione dell'albero decisionale è un fattore critico, ad esempio per risparmiare memoria su dispositivi embedded [8].

Un recente contributo ha portato a un nuovo metodo chiamato MurTree (MT) [9], il quale permette l'elaborazione di un albero decisionale che minimizza il numero di errori di classificazione (misclassification), avendo vincoli sulla profondità e sul numero di nodi. Tuttavia, questo algoritmo non si basa su risolutori a programmazione intera mista (MIP), bensì sulla programmazione dinamica, anche se il problema matematico centrale consenta entrambe le approcci. MT è un algoritmo significativamente più veloce e scalabile rispetto agli algoritmi precedenti per la costruzione di alberi decisionali ottimali. Inoltre, è in grado di produrre alberi decisionali più accurati, in particolare su dataset di grandi dimensioni.

Nel seguito di questo documento, verrà approfondita la letteratura esistente sui modelli di alberi decisionali nel capitolo 2. Questo capitolo presenterà una revisione approfondita degli algoritmi, dei concetti chiave e delle ricerche precedenti riguardanti CART, OCT, e MT, fornendo un contesto completo per la valutazione

comparativa di questi algoritmi.

Il capitolo 3 costituisce invece il nucleo centrale del presente studio. Verrà, quindi, eseguita un'analisi dettagliata e un confronto delle performance e della complessità dei tre algoritmi. Si forniranno tabelle e grafici esplicativi che illustrano le prestazioni di accuratezza dei Decision Trees (DTs) generati, insieme a un'analisi della complessità dei modelli prodotti da CART, OCT, e MT. Inoltre, si dedicherà una sezione specifica al confronto dei tempi di induzione di tali alberi decisionali.

Infine, il capitolo 4 riepiloga e sintetizza i risultati ottenuti dall'analisi comparativa dei tre algoritmi, evidenziando le principali conclusioni tratte e fornendo possibili sviluppi futuri su questo argomento.

Capitolo 2

Stato dell'arte

Come anticipato nel capitolo 1, il presente lavoro si propone di esaminare e confrontare tre algoritmi di induzione di alberi decisionali (DTs, Decision Trees) : Classification and Regression Trees (CART), Optimal Classification Trees (OCT) e MurTree (MT). Prima di procedere con l'analisi dettagliata di questi algoritmi, verrà presentato lo stato dell'arte relativo agli alberi decisionali e ai metodi utilizzati per la loro creazione.

2.1 Alberi Decisionali

I Decision Trees (DTs) sono uno dei principali modelli predittivi di machine learning, e riscuotono un certo successo e interesse per la loro facilità di comprensione e interpretabilità. Un albero decisionale può essere impiegato per attività sia di classificazione che di regressione. I DTs non sono solo un importante strumento di machine learning, ma trovano anche utilizzo in ricerca operativa, in particolare nell'analisi decisionale [10]. Inoltre, sono ampiamente utilizzati in settori come la diagnosi medica, la finanza, l'elaborazione del linguaggio naturale e i sistemi di raccomandazione.

L'induzione dell'albero decisionale comporta la costruzione di una struttura gerarchica in cui ogni nodo interno rappresenta una regola di decisione o una condizione, e ogni nodo foglia corrisponde a un risultato specifico o a un'etichetta di classe [11]. Questa struttura gerarchica facilita la presa di decisioni e fornisce modelli interpretabili, rendendoli una scelta interessante. Infatti, in questo modo le decisioni prese sono giustificate. Segue una panoramica del processo tipico di elaborazione di un DTs:

- **Raccolta dei dati:** è necessario in prima istanza raccogliere un insieme di dati di addestramento, questi dati devono includere esempi di input (caratteristiche) e output (etichette o classi);

- **Pre-elaborazione dei dati:** i dati di addestramento possono richiedere alcune operazioni di pre-elaborazione per renderli puliti e adatti all'induzione del modello. Questo può includere la rimozione di valori mancanti, la normalizzazione delle caratteristiche o la gestione di dati categorici.

- **Creazione del modello:** l'albero decisionale è un tipo di modello di apprendimento automatico. Esistono vari algoritmi di apprendimento che possono essere utilizzati per costruire un albero decisionale, come l'algoritmo ID3, C4.5, CART, e così via. Ognuno di questi algoritmi ha i propri criteri di suddivisione per decidere quale caratteristica utilizzare in ogni passo.

- **Addestramento del modello:** utilizzando i dati di addestramento, l'algoritmo di apprendimento automatico costruisce l'albero decisionale suddividendo le caratteristiche in base ai criteri specificati da questo. Ciò continua fino a quando viene soddisfatto un certo criterio di terminazione (ad esempio, una certa profondità massima dell'albero o un numero minimo di campioni in una foglia).

- **Validazione e ottimizzazione:** a seguito della creazione del DT, è importante valutarne le prestazioni. Questo può essere fatto utilizzando un insieme di dati di validazione separati o attraverso tecniche come la cross-validazione. Se l'albero non ha prestazioni soddisfacenti, possono essere necessari aggiustamenti come la regolazione degli iperparametri o la raccolta di più dati.

- **Test e valutazione finale:** una volta che l'albero decisionale è stato addestrato e ottimizzato, può essere testato su un insieme di dati di test completamente separato che non è stato utilizzato durante l'addestramento o la validazione. Questo fornisce una valutazione finale delle prestazioni del modello.

- **Utilizzo in produzione:** se l'albero decisionale soddisfa le aspettative di prestazioni, può essere utilizzato per prendere decisioni in un ambiente di produzione.

Nella fase di pre-elaborazione dei dati del capitolo 3 verranno utilizzate due tecniche per la puliza e adattamento dei dati, ovvero la one-hot encoding e la label-encoding.

One-Hot Encoding : Il one-hot encoding consiste nella trasformazione di variabili categoriche in un formato numerico adatto all'addestramento di modelli. Nel contesto del one-hot encoding, una variabile categorica con n categorie uniche viene convertita in n nuove colonne binarie. Ciascuna categoria diventa una colonna distinta e ogni istanza assume il valore 1 nella colonna corrispondente alla sua categoria, mentre le altre colonne sono riempite con valori 0. Ad esempio, consideriamo una variabile categorica "Colore" con le variabili "Rosso", "Blu" e "Verde". Il one-hot encoding creerebbe un nuovo insieme di colonne come segue in tabella 2.1:

Tabella 2.1: Esempio di One-Hot Encoding su categoria "Colore"

Colore_Rosso	Colore_Blu	Colore_Verde
1	0	0
0	1	0
0	0	1
1	0	0

Questo metodo permette agli algoritmi di machine learning di lavorare con dati categorici convertendoli in una forma numerica senza introdurre un'errata gerarchia o un ordine tra le categorie. Inoltre, è particolarmente utile quando si desidera evitare ambiguità o interpretazioni erronee tra le variabili categoriche nell'ambito dell'addestramento dei modelli.

Label Encoding : Il Label Encoding è una tecnica di codifica che opera assegnando a ciascuna categoria un numero intero univoco, trasformando così le categorie in valori numerici. Consideriamo, ad esempio, una variabile categorica "Colore" con le variabili "Rosso", "Blu" e "Verde". Utilizzando il Label Encoding, potremmo assegnare loro valori numerici come 0, 1 e 2 come si può vedere in tabella 2.2:

Tabella 2.2: Esempio di Label Encoding su categoria "Colore"

Colore
0
1
2
0

Tuttavia, è importante notare che il Label Encoding potrebbe introdurre un'errata gerarchia o un ordine tra le categorie, considerando i numeri assegnati come rappresentanti di un ordine sequenziale. Questo potrebbe portare a interpretazioni sbagliate da parte degli algoritmi di machine learning, soprattutto se non esiste un ordine intrinseco tra le variabili categoriche. Ciò nonostante, generalmente, la

label encoding è preferibile alla one hot encoding per il training degli alberi decisionali. Questo perché i DTs imparano la relazione tra le variabili indipendenti e la variabile dipendente attraverso una serie di regole if-then. La label encoding consente agli alberi decisionali di imparare queste regole in modo più efficiente rispetto alla one hot encoding.

Come detto in precedenza i modelli possono essere indotti da vari algoritmi, i quali utilizzano diversi criteri per la scelta della caratteristica impiegata nei nodi decisionali dell'albero. I parametri più utilizzati sono l'indice di Gini, l'entropia e il tasso d'errore. I primi due sono, solitamente, adoperati per guidare l'induzione stessa dell'albero, mentre il terzo è utilizzato maggiormente per l'ottimizzazione dei DTs nel processo di potatura.

L'indice di Gini: Una delle più comuni metriche di split dei DTs è l'indice di Gini, utilizzato in particolare da CART, permette di quantificare l'impurità. Il criterio è calcolato considerando la somma dei quadrati delle proporzioni di ciascuna classe all'interno di un nodo:

$$G = 1 - \sum_{i=1}^{k} p_i^2$$

Dove:

- G rappresenta l'Indice di Gini per un nodo.
- k rappresenta il numero di classi nel nodo.
- p_i rappresenta la proporzione della classe i rispetto al totale delle istanze nel nodo.

Nel processo di costruzione dell'albero, l'algoritmo cerca di selezionare la variabile e il punto di divisione che minimizzano l'Indice di Gini. Questo significa che durante la suddivisione dei nodi, il metodo tenta di separare i dati in modo tale che ciascun nodo figlio abbia una distribuzione delle classi il più omogenea possibile, riducendo così l'impurità. Quando l'indice di Gini in un nodo di un albero decisionale è uguale a zero, significa che tutte le istanze presenti in quel nodo appartengono alla stessa classe. Questo scenario rappresenta la massima purezza o omogeneità possibile in quel nodo dell'albero.

L'entropia: L'entropia è un altro criterio comunemente utilizzato nella costruzione degli alberi decisionali, questa metrica valuta l'impurità o l'incertezza all'interno di un nodo, infatti, esprime la misura di disordine nel contesto della distribuzione delle classi dei dati. Nell'ambito degli alberi decisionali, l'entropia di un nodo è calcolata come:

$$H = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

Dove:

- H rappresenta l'entropia per un nodo.
- k rappresenta il numero di classi nel nodo.
- p_i rappresenta la proporzione della classe i rispetto al totale delle istanze nel nodo.

Durante la costruzione dell'albero, l'obiettivo è quello di massimizzare la riduzione dell'entropia, cercando di trovare la suddivisione dei nodi che minimizza l'incertezza, separando i dati in modo da ottenere nodi figlio più omogenei possibile rispetto alla variabile target. Se l'entropia di un nodo è pari a zero, significa che

tutte le istanze presenti in quel nodo appartengono alla stessa classe, indicando la massima purezza in quel particolare nodo dell'albero decisionale. In pratica, ciò suggerisce che non è necessario ulteriormente suddividere quel nodo poiché contiene solo istanze di una singola classe.

Tasso d'errore: Il tasso d'errore o misclassificazione è un altro criterio utilizzato per valutare la purezza dei nodi negli alberi decisionali. Questa metrica misura l'errore di classificazione all'interno di un nodo, considerando la proporzione della classe maggioritaria. Nel contesto degli alberi decisionali, il tasso d'errore di un nodo è calcolato come:

$$\text{Tasso d'errore} = 1 - \max(p_i)$$

Dove:

- Il *Tasso d'errore* rappresenta l'errore di classificazione per un nodo.
- p_i è la proporzione della classe i più comune rispetto al totale delle istanze nel nodo.

Durante la costruzione dell'albero, si cerca di ridurre al minimo il tasso d'errore dei nodi, scegliendo divisioni che permettano di ottenere nodi figlio con una distribuzione delle classi più omogenea possibile. Un nodo con una misclassificazione pari a zero indica che tutte le istanze presenti sono state classificate correttamente, questo suggerisce che non è necessario ulteriormente suddividere quel nodo in quanto contiene solo istanze di una singola classe.

Nell'induzione dei DTs è importante sapere che un albero più piccolo riesce più facilmente a raggiungere nodi foglia semplici, al contrario con il crescere di dimensione dell'albero diventa più difficile mantenere questa semplicità dando origine a

un fenomeno noto come frammentazione dei dati. Questo evento si riferisce alla suddivisione di un insieme di dati in porzioni più ridotte, spesso portando ad un overfitting, dove l'albero si adatta troppo alle caratteristiche dell'insieme di addestramento e non generalizza bene i nuovi dati. Per questa ragione, i DTs prediligono strutture piccole che spesso si ottengono tramite l'utilizzo di tecniche di potatura (pruning). In questo contesto individuiamo due principali tipi di potatura: la prima chiamata pre-pruning che limita la crescita dell'albero durante l'addestramento impedendo, quindi, l'espansione quando si verifica una determinata condizione, come potrebbe essere uno scarso numero di campioni o il raggiungimento di una certa profondità. L'altro tipo di potatura è il post-pruning, dove l'albero viene addestrato fino a raggiungere la sua massima profondità, e alcune delle foglie vengono unite o rimosse. Ciò viene svolto tramite criteri che possono essere basati su metriche di valutazione delle prestazioni, come l'accuratezza o l'entropia, applicate ai nodi foglia dell'albero.

Gli alberi decisionali sono particolarmente vantaggiosi rispetto ad altri metodi di apprendimento automatico, non solo per la loro facilità di comprensione, ma anche per la loro capacità di essere addestrati su insiemi di dati sia categorici che continui. Questa versatilità li rende strumenti potenti per la creazione di modelli predittivi in diversi contesti. Un aspetto chiave dei DTs è la loro flessibilità: non si limitano alla classificazione binaria, ma supportano anche la multi-classificazione, consentendo l'assegnazione di più etichette o classi a un'istanza. Mentre la classificazione binaria fornisce un'etichettatura dicotomica, come vero o falso, 0 o 1, la multi-classificazione permette di assegnare più classi o categorie a un singolo esempio. Inoltre, gli alberi decisionali sono in grado di gestire la regressione, un'area diversa rispetto alla classificazione, che mira a predire un valore numerico piuttosto che un'etichetta di classe. Questo significa che, mentre la classificazione assegna un'etichetta a un'istanza in base alle sue caratteristiche, la regressione

cerca di predire un valore numerico, come il prezzo di una casa o la previsione di una variabile continua.

In sintesi, la potenza dei DTs risiede nella loro capacità di eseguire la classificazione binaria, la multi-classificazione e la regressione, offrendo così un ampio spettro di applicazioni in diversi domini e contesti di apprendimento automatico.

D'altro canto, sono presenti anche degli svantaggi, in quanto gli alberi decisionali sono spesso inclini all'overfitting e la loro induzione richiede costi computazionali più costosi rispetto che ad altri algoritmi di machine learning.

2.2 Classification and Regression Trees (CART)

In questa sezione verrà presentato uno dei metodi di induzione di alberi decisionali più popolari e potenti: l'algoritmo Classification and Regression Trees (CART).

2.2.1 Algoritmo CART

L'algoritmo CART è stato sviluppato da Breiman et al. nel 1984 [1], esso è ampiamente utilizzato per la costruzione di alberi decisionali nelle applicazioni di classificazione e regressione. Questo metodo opera partizionando in modo ricorsivo i dati di addestramento in sottoinsiemi più piccoli utilizzando suddivisioni binarie.

Il metodo CART riscuote notevole successo grazie alla sua capacità di gestire sia caratteristiche categoriche che continue, alla sua interpretabilità e alla sua abilità di catturare relazioni non lineari tra le caratteristiche e la variabile target. Queste aspetti rendono CART flessibile e adatto a una vasta gamma di problemi.

L'algoritmo Classification and Regression Trees (CART) induce sempre un albero binario, ciò è in contrasto con altri metodi basati su alberi, che potrebbero consentire nodi figli multipli. L'albero inizia dal nodo radice, che contiene tutti i

dati di addestramento, e procede a suddividere in modo ricorsivo i dati in sottoinsiemi più piccoli fino a quando non viene soddisfatto un criterio di arresto (come la profondità massima per l'albero o il numero minimo di istanze in ciascun nodo foglia).
Ad ogni nodo dell'albero, l'algoritmo seleziona una caratteristica e una soglia che separano al meglio i dati di addestramento in due gruppi, basandosi sui valori di quella caratteristica. Questa selezione avviene utilizzando il criterio di Gini o altri criteri come l'entropia.

Una volta che l'albero è stato costruito, è possibile utilizzarlo per effettuare previsioni, navigando attraverso di esso dal nodo radice a un nodo foglia in base ai dati di test corrispondenti.

CART può gestire diversi tipi di diversi tipi di variabili sia per le caratteristiche (variabili indipendenti, X) che per la variabile obiettivo (variabile dipendente, y). I tipi di variabili che CART riesce a gestire in input sono:

- **Variabili continue:** Le variabili continue rappresentano misurazioni su una scala continua (ad esempio, altezza, peso, temperatura). L'algoritmo CART può gestire variabili continue senza problemi.

- **Variabili categoriche:** Queste variabili rappresentano categorie discrete senza un ordine intrinseco (ad esempio, colore, genere, tipo di veicolo). L'algoritmo CART può gestire variabili categoriche ma richiede la codifica in numeri interi per poterle utilizzare nell'addestramento tramite, ad esempio, le tecniche viste precedentemente: One-Hot Encoding e Label Encoding.

- **Variabili discrete:** Le variabili che assumono un numero finito di valori, l'algoritmo si basa principalmente su queste e le gestisce senza problemi.

- **Variabili binarie:** Sono variabili discrete a due valori (0 e 1), il metodo CART le gestisce bene, ma se l'insieme di dati è vasto e con un altro numero di caratteristiche, il modello generato potrebbe sovraaddattarsi.

Detto ciò, l'algoritmo CART ha una grande flessibilità sulle variabili, infatti, può trattare dati con valori mancanti, ma spesso richiede una gestione preventiva dei dati mancanti durante la fase di pre-processing.

Figura 2.1: Esempio di albero decisionale indotto tramite l'algoritmo CART sull'insieme di dati 'nath-jones' di 46 esempi.

La fig. 2.1 mostra un albero decisionale ottenuto mediante l'impiego dell'algoritmo Classification and Regression Trees (CART) sull'insieme di dati 'nath-jones', composto da 46 esempi. Questo dataset presenta quattro caratteristiche (CF/TD,

NI/TA, CA/CL, CA/CS) con valori continui, conducenti ad un'etichetta di classe binaria. L'algoritmo CART utilizza l'indice di Gini come criterio per effettuare suddivisioni ricorsive del dataset, limitando la profondità massima a 4 livelli. Tuttavia, è possibile notare che l'undicesimo nodo avrebbe potuto essere ulteriormente suddiviso in due nodi figli.

2.2.2 Vantaggi e Svantaggi

Vantaggi:

- È un algoritmo semplice ed intuitivo, facile da comprendere e interpretare;
- Possibilità di scegliere il criterio di divisione tra indice di Gini, Entropia e Log Loss (logaritmo della perdita).
- Può gestire sia dati numerici che categorici;
- Può gestire i valori mancanti imputandoli con suddivisioni surrogate;
- Può gestire problemi di classificazione multiclasse.

Svantaggi:

- Tende a sovraadattare i dati, specialmente se l'albero è lasciato crescere troppo in profondità;
- È un algoritmo greedy che potrebbe non trovare l'albero ottimale;
- Potrebbe produrre risultati instabili se i dati sono sensibili a piccoli cambiamenti o rumore;

In conclusione, l'algoritmo CART rappresenta una potente tecnica per la costruzione di alberi decisionali interpretabili e può essere applicato con successo a una vasta gamma di problemi di machine learning.

2.3 Optimal Classification Trees (OCT)

In questa sezione, esamineremo gli Alberi di Classificazione Ottimali (OCT, Optimal Classification Trees) [4], come proposti da Bertsimas e Dunn nel loro lavoro del 2017.

2.3.1 Algoritmo OCT

Gli Alberi di Classificazione Ottimali (OCT) rappresentano un avanzato metodo per costruire alberi decisionali che minimizzano l'errore di classificazione su un set di dati di addestramento.

L'approccio OCT proposto da Bertsimas e Dunn si basa su un problema di ottimizzazione matematica che cerca di trovare la miglior divisione dei dati di addestramento al fine di massimizzare l'accuratezza di classificazione. Benché l'induzione di alberi decisionali sia noto come un problema NP-Hard fin dalle origini dell'algoritmo CART [3] [1], all'epoca non era computazionalmente possibile raggiungere una soluzione ottima. Tuttavia, grazie alle tecniche di ottimizzazione miste intere (MIO), oggi è possibile mirare alla migliore soluzione.

Il metodo Optimal Classification Trees (OCT) utilizza un algoritmo di programmazione lineare per costruire l'intero albero decisionale contemporaneamente. L'algoritmo tiene conto di tutte le possibili combinazioni di split per trovare la soluzione che minimizza l'errore di classificazione. La soluzione viene ottimizzata tramite un insieme di dati di validazione e successivamente si rimuovono i nodi non necessari dall'albero decisionale per ridurre l'overfitting.

Il metodo di induzione OCT porta alla creazione di alberi con un minor numero di nodi rispetto agli algoritmi euristici che lo precedono, risultando in una miglior leggibilità e una maggiore accuratezza.

Questo algoritmo può gestire diversi tipi di variabili sia per le caratteristiche che per la classficazione. I tipi di variabili che OCT è in grado di gestire in input includono:

- **Variabili continue:** Queste rappresentano misurazioni su una scala continua, ad esempio altezza, peso e temperatura. OCT può gestire variabili continue senza problemi.
- **Variabili categoriche:** Queste rappresentano categorie discrete senza un ordine intrinseco, come colore, genere e tipo di veicolo. OCT può gestire variabili categoriche, ma richiede la codifica in numeri interi per l'addestramento. Tecniche come il One-Hot Encoding o il Label Encoding sono comuni per questa codifica.
- **Variabili discrete:** Si riferiscono a variabili che assumono un numero finito di valori. Anche per questo tipo di variabile l'algoritmo non riscontra difficoltà.
- **Variabili binarie:** Le variabili binarie, che assumono solo due valori (0 o 1), sono comunemente utilizzate in OCT e possono essere gestite direttamente dall'algoritmo senza la necessità di ulteriori manipolazioni.

OCT possiede notevole flessibilità nella gestione delle variabili delle caratteristiche e può trattare dati con valori mancanti. Tuttavia, spesso richiede una gestione preventiva di tali dati mancanti durante la fase di pre-processing.

La fig. 2.2 mostra un albero decisionale indotto tramite l'uso dell'algoritmo Optimal Classification Trees (OCT), in particolare un'implementazione della libreria Python: 'interpretableai' [12]. Il metodo OCT utilizza il tasso d'errore di classificazione, come criterio di ottimizzazione, per creare l'albero decisionale sull'insieme di dati 'nath-jones' avente 46 esempi con valori continui; in questo caso non è necessaria alcuna pre-elaborazione dei dati, siccome l'algoritmo è in grado di gestire

questi. Vi è inoltre, un vincolo di profondità massima di 4. Si può notare come il DT, non sempre, presenta il massimo di purezza nei nodi foglia, e raggiunga una profondità con valore 2, questo sintomo di una ottimizzazione dell'albero tramite tecniche di potatura per una miglior generalizzazione su insiemi di dati non noti.

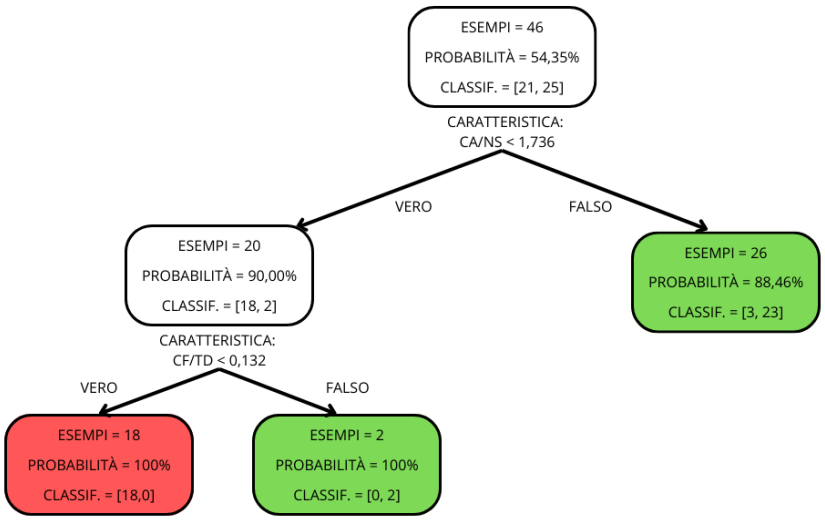


Figura 2.2: Esempio di albero decisionale indotto tramite l'algoritmo OCT sull'insieme di dati 'nath-jones' di 46 esempi.

2.3.2 Vantaggi e Svantaggi

Vantaggi:

- Capacità di creare alberi con buona accuratezza di presivisione ottimizzata per specifici criteri di classificazione;
- Possibilità di scegliere il criterio di divisione tra Misclassification, indice di Gini e Entropia.
- Maggiore flessibilità nella scelta dei criteri di performance rispetto ad altri algoritmi di alberi decisionali;

- Riduzione del rischio di overfitting grazie all'approccio di ottimizzazione globale;
- Maggiore interpretabilità rispetto ai tradizionali alberi decisionali.

Svantaggi:

- Potenziale complessità computazionale elevata, specialmente per dataset di grandi dimensioni o con molte caratteristiche;
- Maggiore consumo di risorse computazionali rispetto agli approcci euristici tradizionali;
- Sensibilità alla qualità dei dati di addestramento e alla rappresentatività del campione;
- Richiede una buona comprensione delle tecniche di ottimizzazione matematica per essere implementato correttamente.

Gli OCT trovano applicazione in una vasta gamma di settori, tra cui la medicina, la finanza, la biologia e l'ingegneria. Sono particolarmente utili in situazioni in cui la precisione di classificazione è di fondamentale importanza.

In conclusione, gli Alberi di Classificazione Ottimali rappresentano una potente estensione degli algoritmi di alberi decisionali, focalizzati sulla massimizzazione delle performance di classificazione. La loro flessibilità e precisione li rendono un'opzione attraente per una varietà di applicazioni di machine learning.

2.4 MurTree (MT)

Il paradigma degli Alberi di Decisione Ottimali (OCT) è esteso da MurTree (MT), un innovativo approccio proposto da Demirović et al. [9]. Nella presente sezione viene descritto questo metodo di induzione di alberi decisionali.

2.4.1 Algoritmo MT

MurTree si differenzia per la sua attenzione alla qualità ottimale degli alberi di decisione. A differenza dei classici metodi OCT che utilizzano tecniche di ottimizzazione miste intere (MIO), l'algoritmo MT si basa su programmazione dinamica e ricerca, combinando bene precisione e efficienza.

L'algoritmo MT inizia costruendo un albero vuoto. Quindi, itera su tutti i possibili nodi decisionali e valuta la qualità di ogni possibile albero che si potrebbe costruire aggiungendo quel nodo. Il metodo sceglie il nodo decisionale che produce il DT con la migliore qualità e lo aggiunge all'albero. Questo processo viene ripetuto fino a quando l'albero non è completo.

L'efficacia di MurTree risiede nell'uso della programmazione dinamica per memorizzare le valutazioni dei possibili alberi, minimizzando il bisogno di ripeterle. La ricerca viene impiegata per individuare il nodo decisionale ottimale, garantendo la massima qualità della soluzione.

MurTree, come proposto da Demirović et al. [9], offre, anche, un approccio all'induzione di alberi di decisione ottimali. Tuttavia, è importante notare che l'implementazione fornita dagli autori è progettata specificatamente per gestire insiemi di dati con variabili binarie. Ciò significa che tutte le variabili, incluse quelle continue, devono essere convertite in variabili binarie prima dell'uso con MurTree. Poiché MT può operare solo con variabili binarie, è necessario eseguire una trasformazione adeguata delle variabili categoriche o continue prima dell'addestramento del modello. Una pratica comune per trasformare le variabili continue in binarie, è quello di convertirle in prima istanza in categoriche, questo si può fare tramite delle primitive fornite dalla libreria arulesCBA [13], e successivamente trasformare le variabili categoriche ottenute in binarie per via di tecniche come la One-Hot Encoding o la Label Encoding. Questa limitazione di MurTree impone la necessità di una manipolazione preliminare dei dati per adattare correttamente

il dataset ai requisiti dell'algoritmo. Solo dopo aver convertito tutte le variabili categoriche in binarie sarà possibile utilizzare l'algoritmo MT per la costruzione di alberi decisionali su tali insieme di dati.

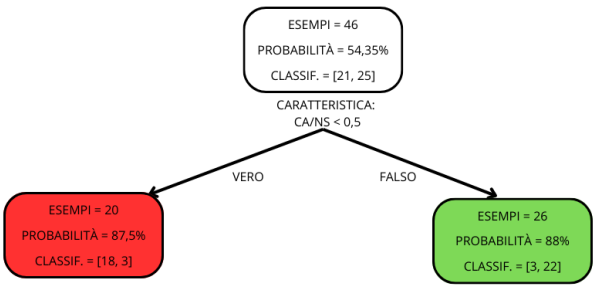


Figura 2.3: Esempio di albero decisionale indotto tramite l'algoritmo MT sull'insieme di dati 'nath-jones' di 46 esempi.

La fig. 2.3 mostra un esempio di albero decisionale elaborato da MT sull'insieme di dati 'nath-jones'. Prima di essere utilizzato dall'algoritmo, il dataset è stato sottoposto a una conversione: le variabili continue sono state trasformate in categoriche e successivamente binarizzate. Questo processo consente a MurTree di operare con successo, considerando solo valori binari (0 o 1). Il metodo viene eseguito con un vincolo di profondità 4, come per la fig. 2.1 e la fig. 2.2, ma in questo caso viene indotto un albero meno complesso, visto che MT applica un'ottimizzazione tramite una forte potatura.

2.4.2 Vantaggi e Svantaggi

Vantaggi:

- Precisione ottimale nella costruzione dell'albero decisionale;
- Efficienza computazionale sugli insiemi di dati di grandi dimensioni;

- Gestione avanzata di dati rappresentativi.

Svantaggi:

- Complessità computazionale potenzialmente elevata;
- Nessuna possibilità di scelta dei criteri di splitting;
- Richiede conoscenza avanzata di ottimizzazione matematica;
- Necessità di caratteristiche con variabili binarie.

L'algoritmo MurTree trova impiego in una varietà di contesti, tra cui la previsione del comportamento d'acquisto dei clienti, la diagnosi medica basata su dati complessi e l'ottimizzazione di processi industriali.

In conclusione, MurTree rappresenta un avanzamento significativo nella costruzione di alberi di decisione, con un focus sull'ottimizzazione globale e la massima precisione.

Capitolo 3

Contributo

Il presente capitolo mostra un'analisi dettagliata e un confronto tra tre algoritmi di induzione di alberi decisionali: Classification and Regression Trees (CART), Optimal Classification Trees (OCT) e MurTree (MT). Ciò viene fatto tramite degli esperimenti computazionali. Prima di entrare nel dettaglio degli esperimenti, vengono riassunte le notazioni scientifiche in tabella 3.1.

Tabella 3.1: Riassunto delle notazioni

Insiemi	
I	Insieme dei dati di addestramento
F	Insieme delle caratteristiche
C	Insieme delle etichette
X	Variabile
Tipi di variabile	
c	Variabile continua
cat	Variabile categorica
d	Variabile discreta
bin	Variabile binaria

3.1 Configurazione degli Esperimenti

Insieme di Dati: La sperimentazione si basa su un insieme di 12 dataset, riassunti nella tabella 3.2 e ordinati in base al numero di esempi. Questi dataset presentano una varietà di caratteristiche, includendo variabili discrete, continue, categoriche e binarie. Alcuni dataset mostrano esempi con dati mancanti, mentre altri presentano variabili target con molteplici etichette.

I tre algoritmi di induzione, CART, OCT, e MT, utilizzano gli stessi dataset. Tuttavia, è importante notare che il metodo MT, come descritto nella sezione 2.4, gestisce solo caratteristiche binarie, richiedendo una codifica tramite One-Hot Encoding nella fase di pre-addestramento. L'utilizzo di variabili binarie potrebbe comportare un significativo aumento del numero di caratteristiche, richiedendo potenzialmente alberi più profondi. Pertanto, per CART e OCT, si utilizzano i dati grezzi, dove possibile, come per le variabili discrete e continue, poiché questi algoritmi sono in grado di gestirli in modo efficace. Mentre, per le variabili categoriche nella fase di pre-addestramento di CART e OCT, viene eseguita una codifica dei dati tramite la tecnica Label Encoding.

Il primo dataset, denominato 'nath-jones' come mostrato nella tabella 3.2, si componi di 46 esempi e presenta 4 caratteristiche con variabili continue e 2 etichette di classificazione binaria (0 o 1). Segue 'soybean-small' con 47 casi e caratterizzato da 35 features con variabili discrete e 4 diverse classificazioni con la seguente distribuzione:

- *D1*: 10
- *D2*: 10
- *D3*: 10
- *D4*: 17

I dataset 'monks-1', 'monks-2' e 'monks-3', rispettivamente con 124, 169 e 122 esempi, presentano 7 caratteristiche discrete e etichettazione binaria, senza attributi con valori mancanti. 'hayes-roth', con 132 esempi, possiede 5 diverse caratteristiche discrete e ciascun caso può essere classificato in 3 diverse categorie. Segue 'house-votes-84' con 16 features categoriche, alcuni dati con valori mancanti, gestiti eliminando le righe prive di valori. Le classi sono due: 'democrat' e 'republican'. Il dataset 'spect' presenta 22 attributi con variabili binarie e classificazione altrettanto (0 o 1), senza valori mancanti. L'insieme di dati 'breast-cancer' include 9 caratteristiche con variabili categoriche; alcuni esempi presentano valori mancanti, gestiti rimuovendo le righe dove assenti. La classificazione è binaria. L'insieme 'balance-scale' contiene 625 esempi con 4 attributi distinti e 3 possibili etichette, senza dati mancanti, tutte le caratteristiche sono discrete. 'tic-tac-toe' include 958 casi con 9 caratteristiche categoriche, senza valori mancanti e con etichettatura binaria. Infine, 'car-evaluation' contiene 1728 esempi con 6 attributi categorici e 4 diverse etichette di classificazione:

- *unacc*: 1210
- *acc*: 384
- *good*: 69
- *v-good*: 65

Anche per quest'ultimo insieme non vi sono casi con valori, mancanti. Tutti i datasets sono riassunnti nella tabella 3.2.

Tabella 3.2: Riassunto degli insiemi di dati

Insieme di Dati	I	F	X	C
nath-jones	46	4	c	2
soybean-small	47	35	d	4
monks-3	122	7	d	2
monks-1	124	7	d	2
hayes-roth	132	5	d	3
monks-2	169	7	d	2
house-votes-84	232	16	cat	2
spect	267	22	bin	2
breast-cancer	277	9	cat	2
balance-scale	625	4	d	3
tic-tac-toe	958	27	cat	2
car-evaluation	1728	6	cat	4

Addestramento del DT: La fase di addestramento dell'albero coinvolge la creazione di cinque suddivisioni casuali distinte per ciascun insieme di dati. Successivamente, ognuna di queste viene partizionata in tre sottoinsiemi: il primo comprende il 50% degli esempi ed è utilizzato per l'allenamento, mentre i restanti due, ciascuno comprendente il 25%, sono dedicati alla fase di validazione e di test. I criteri di arresto dell'addestramento sono la profondità massima, la quale cambia per ogni esperimento, partendo inizialmente da 2 e arrivando a 5, o nel peggiore dei casi, esiste un limite di tempo di 10 minuti.

L'algoritmo CART è stato implementato mediante l'utilizzo della libreria scikit-learn [14]. Per quanto riguarda OCT, è stato utilizzato il framework di Intrepretable AI [12]. Infine, l'algoritmo Murtree è stato implementato seguendo l'approccio di Emir Demirović et al. [9]. Diversamente da CART e OCT, che eseguono su Python, Murtree è scritto in C++ e non fornisce delle interfacce per accedere al-

la struttura dati contenente l'albero decisionale. Nel prossimo paragrafo segue il processo di standardizzazione degli alberi decisionali indotti.

Standardizzazione dei DTs: Per il corretto sviluppo degli esperimenti, è stato necessario standardizzare gli alberi decisionali generati dai tre algoritmi: Classification and Regression Trees (CART), Optimal Classification Trees (OCT), e MurTree (MT). Poiché CART è implementato tramite la libreria scikit-learn [14] in Python, OCT utilizza delle primitive che eseguono in linguaggio Julia su ambiente Python [12] e MT è implementato in C++, si è reso essenziale ottenere una struttura dati comune per tutti e tre gli algoritmi.

CART e OCT, seppur forniscano delle interfacce, a diferrenza di MT, queste utilizzano indici diversi e nomi differenti, quindi è stata eseguita una ricerca in ampiezza (BFS) attraverso la struttura dati degli alberi decisionali. Questo processo ha permesso di esplorare tutti i nodi e di salvare la struttura dati risultante in formato JSON tramite le funzione presenti nel listato 3.1 e nel listato 3.2. Per MT, invece, è stato necessario modificare il file "main.cpp" creando nuove interfacce e aggiungendo la libreria nlohmann/json [15], al fine di salvare gli alberi decisionali, sempre in un formato comune ai tre algoritmi. Il listato 3.3 mostra una funzione in codice C++ che esegue la conversione dell'albero indotto da MT in formato JSON.

Listato 3.1: Funzione in codice Python utilizzata per esplorare un albero decisionale indotto tramite CART e salvare la struttura dati in formato JSON

```
import json
def convert_cart_to_json(tree, node_index=0, depth=0):
    json_tree = {}
    if tree.children_left[node_index] == tree.children_right[node_index]:
        json_tree["type"] = "leaf"
        json_tree["num_samples"] = int(tree.n_node_samples[node_index])
        json_tree["class_probabilities"] = tree.value[node_index].tolist()[0]
        json_tree["label"] = int(tree.value[node_index].argmax())
```

```
        json_tree["depth"] = int(depth)
        json_tree["node_index"] = int(node_index)
    else:
        json_tree["type"] = "split"
        json_tree["split_feature"] = int(tree.feature[node_index])
        json_tree["split_threshold"] = float(tree.threshold[node_index])
        json_tree["depth"] = int(depth)
        json_tree["node_index"] = int(node_index)
        json_tree["left_child"] = convert_cart_to_json(tree, tree.children_left[
            node_index], depth + 1)
        json_tree["right_child"] = convert_cart_to_json(tree, tree.children_right[
            node_index], depth + 1)

    return json_tree
```

Listato 3.2: Funzione in codice Python utilizzata per esplorare un albero decisionale indotto tramite OCT e salvare la struttura dati in formato JSON

```
import json
def convert_oct_to_json(tree, node_index=1, depth=0):
    json_tree = {}
    if tree.is_leaf(node_index):
        json_tree["type"] = "leaf"
        json_tree["num_samples"] = tree.get_num_samples(node_index)
        json_tree["class_probabilities"] = tree.get_classification_proba(
            node_index)
        json_tree["label"] = tree.get_classification_label(node_index)
        json_tree["depth"] = depth
        json_tree["node_index"] = node_index-1
    else:
        json_tree["type"] = "split"
        json_tree["split_feature"] = int(tree.get_split_feature(node_index))-1
        json_tree["split_threshold"] = tree.get_split_threshold(node_index)
        json_tree["depth"] = depth
        json_tree["node_index"] = node_index-1
        json_tree["left_child"] = convert_oct_to_json(tree, tree.get_lower_child(
            node_index), depth + 1)
        json_tree["right_child"] = convert_oct_to_json(tree, tree.get_upper_child(
            node_index), depth + 1)

    return json_tree
```

Listato 3.3: Funzione in codice C++ utilizzata per esplorare un albero decisionale indotto tramite MT e salvare la struttura dati in formato JSON

```
json convertTreeToJson(MurTree::DecisionNode* node, int depth = 0, int node_index
    = 0) {
    json jsonNode;

    if (node->IsLabelNode()) {
        jsonNode["type"] = "leaf";
        jsonNode["label"] = node->label_;
    jsonNode["depth"] = depth;
    } else {
        jsonNode["type"] = "split";
        jsonNode["split_feature"] = node->feature_;
    jsonNode["depth"] = depth;
    jsonNode["node_index"] = node_index;
        jsonNode["left_child"] = convertTreeToJson(node->left_child_, depth + 1, 2
            * node_index + 1);
        jsonNode["right_child"] = convertTreeToJson(node->right_child_, depth + 1,
            2 * node_index + 2);
    }

    return jsonNode;
}
```

Come evidenziato nella fig. 3.1, che mostra un esempio di struttura dati JSON rappresentante un albero decisionale addestrato sull'insieme di dati 'nath-jones' tramite il metodo OCT, si nota una standardizzazione del modello predittivo. Gli alberi decisionali presentano nodi di due tipologie: "split" o "leaf". I nodi "split" sono nodi di divisione che generano due nodi figli, mentre i nodi "leaf" sono nodi foglia. I nodi "split" contengono informazioni sull'indice della caratteristica utilizzata per la divisione, indicata come "split_feature" e sulla soglia dei relativi valori "split_threshold". Nel caso specifico, il nodo di suddivisione è relativo alla caratteristica posizionata nella prima colonna del dataset con soglia dei valori

0.07409. Al contrario, i nodi "leaf" includono informazioni sull'etichettatura della classificazione, indicata come "label", il numero di esempi associati a quel nodo e altre descrizioni pertinenti.


```
{
  "type": "split",
  "split_feature": 0,
  "split_threshold": 0.07409999999999994,
  "depth": 0,
  "node_index": 0,
  "left_child": {
    "type": "leaf",
    "num_samples": 11,
    "class_probabilities": {
      "0": 0.9090909090909091,
      "1": 0.09090909090909091
    },
    "label": 0,
    "depth": 1,
    "node_index": 1
  },
  "right_child": {
    "type": "leaf",
    "num_samples": 11,
    "class_probabilities": {
      "0": 0,
      "1": 1
    },
    "label": 1,
    "depth": 1,
    "node_index": 2
  }
}
```


Figura 3.1: Esempio di struttura dati JSON che rappresenta un albero decisionale sull'insieme di dati 'nath-jones' indotto dall'algoritmo OCT.

Una volta trovata una struttura dati comune, è stato necessario scrivere una funzione in codice Python che potesse classificare tramite l'albero decisionale salvato in formato JSON i vari esempi. Questo è stato possibile tramite la funzione "predict" che si trova scritta nel listato 3.4. Questa funzione, dati un albero decisionale e un esempio, ripercorre il primo dal nodo radice raggiungendo un nodo foglia contente un'etichetta di classificazione del caso dato.

Listato 3.4: Funzione in codice Python utilizzata classificare un esempio tramite un albero decisionale

```
from collections import deque
def predict(tree, x):
```

```
    queue = deque()
    queue.append(tree)
    while len(queue) != 0:
        node = queue.popleft()
        if node['type'] == 'split':
            if float(x[node['split_feature']]) <= node['split_threshold']:
                queue.append(node['left_child'])
            else:
                queue.append(node['right_child'])
        else:
            return node['label']
```

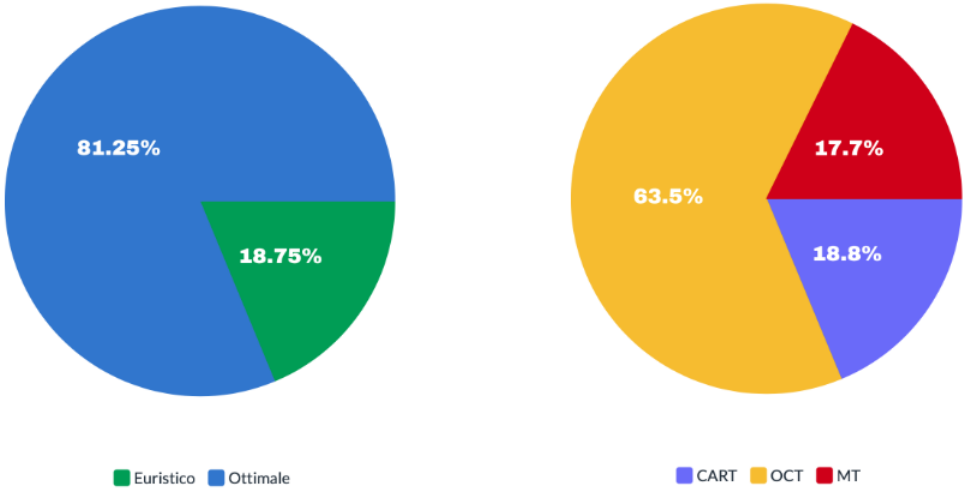
Metodologie di test della classficazione dei DTs: Per valutare le prestazioni degli alberi decisionali indotti da CART, OCT e MT, è stata condotta una serie di test su insiemi di dati diversificati. Per i vari esperimenti con diverse profondità massime, è stata calcolata la media delle percentuali di classificazioni corrette sui dati di test non visti dall'albero decisionale. Questa approfondita analisi delle performance aiuta a fornire una visione completa delle capacità predittive di ciascun algoritmo, considerando vari livelli di complessità degli alberi decisionali.

Ambiente di Test: Tutti gli esperimenti sono stati condotti utilizzando un processore Intel Core i3 dual-core a 1,1 GHz, con a disposizione una memoria RAM di 8GB LPDDR4X a 3.733MHz. Importante precisare che le implementazioni di CART e di OCT sono state fatte in Python, mentre quella di MT in C++, il che gli conferisce un sostanziale vantaggio in termini di tempo di esecuzione, in quanto quest'ultimo impiega mediamente, a parità di routine, meno tempo per giungere a termine [16].

3.2 Accuratezza di previsione dei DTs indotti:

L'obiettivo è analizzare quale albero decisionale presenta la maggiore accuratezza nella previsione degli esempi di test. Tale valutazione fornirà informazioni cruciali per comprendere le prestazioni dei modelli predittivi indotti da CART, OCT, e MT nel contesto specifico dell'insieme di dati considerati.

Come si può notare dalla tabella 3.3 gli algoritmi ottimali forniscono un albero decisionale con più alta percentuale di accuratezza sull'insieme di dati di test nella maggior parte degli esempi. Questo a dimostrazione che gli algoritmi euristici, come CART, tendono a sovradattare il modello classificando un gruppo di dati non visto in modo più impreciso. Infatti, gli algoritmi ottimali (OCT e MT) classificano con miglior accuratezza 39 volte su 48, come si evince dalla fig. 3.2a con una percentuale di 81,25% contro il 18,75% degli algoritmi euristici (in questo caso CART).



(a) Algoritmi euristici e ottimali (b) CART, OCT, MT

Figura 3.2: Percentuale di volte in cui gli algoritmi sono risultati il più accurati nell'insieme di dati, basato su svariati esperimenti.

Tabella 3.3: Accuratezza di previsione media di CART, OCT, MT - Parte 1

Insieme di Dati	prof.tà massima	CART	OCT	MT
nath-jones	2	0.8167	0.7500	**0.8333**
nath-jones	3	**0.8500**	0.7500	**0.8500**
nath-jones	4	0.8333	0.7500	**0.8500**
nath-jones	5	0.8167	0.7500	**0.8500**
soybean-small	2	0.7500	**1.0000**	0.9667
soybean-small	3	**1.0000**	**1.0000**	0.9333
soybean-small	4	0.9833	**1.0000**	0.9500
soybean-small	5	0.9500	**1.0000**	0.8667
monks-3	2	**0.9727**	0.9684	0.9290
monks-3	3	0.9785	**0.9914**	0.8645
monks-3	4	0.9899	**0.9914**	0.7613
monks-3	5	0.9871	**0.9914**	0.7290
monks-1	2	0.7381	0.7159	**0.7613**
monks-1	3	0.7899	0.8174	**0.8645**
monks-1	4	0.7842	**1.0000**	0.9613
monks-1	5	0.7597	**1.0000**	0.8645
hayes-roth	2	0.4950	**0.5650**	0.4848
hayes-roth	3	0.5550	**0.6200**	0.5879
hayes-roth	4	0.6300	**0.7800**	0.6364
hayes-roth	5	0.7550	**0.7600**	0.6606
monks-2	2	**0.6106**	0.6000	0.6000
monks-2	3	0.5921	**0.6560**	0.6476
monks-2	4	0.6040	**0.6427**	0.5381
monks-2	5	**0.6755**	0.6480	0.5429

Tabella 3.3: Accuratezza di previsione media di CART, OCT, MT - Parte 2

Insieme di Dati	prof.tà massima	CART	OCT	MT
house-votes-84	2	0.9759	**0.9828**	0.9621
house-votes-84	3	0.9655	**0.9759**	0.9345
house-votes-84	4	0.9621	**0.9759**	0.9138
house-votes-84	5	0.9655	**0.9759**	0.9345
spect	2	0.7373	**0.8030**	0.7791
spect	3	0.7403	**0.8030**	0.7552
spect	4	0.7463	**0.8000**	0.7224
spect	5	0.7463	**0.8091**	0.7284
breast-cancer	2	**0.7400**	0.7114	0.7014
breast-cancer	3	0.7143	**0.7286**	0.6928
breast-cancer	4	0.7029	**0.7171**	0.6754
breast-cancer	5	0.6886	**0.7229**	0.6261
balance-scale	2	0.6268	**0.6586**	0.6436
balance-scale	3	0.7147	**0.7287**	0.6885
balance-scale	4	**0.7694**	0.7478	0.7090
balance-scale	5	**0.7631**	0.7529	0.6962
tic-tac-toe	2	0.6992	**0.7108**	0.6867
tic-tac-toe	3	0.7183	0.7458	**0.7500**
tic-tac-toe	4	0.7808	0.7825	**0.8233**
tic-tac-toe	5	0.7942	0.8242	**0.8817**
car-evaluation	2	**0.7815**	0.7714	0.7796
car-evaluation	3	0.7907	**0.7958**	0.7847
car-evaluation	4	**0.8482**	0.8416	0.8097
car-evaluation	5	0.8444	**0.8642**	0.8616

3.3 Complessità dei DTs indotti da CART, OCT e MT:

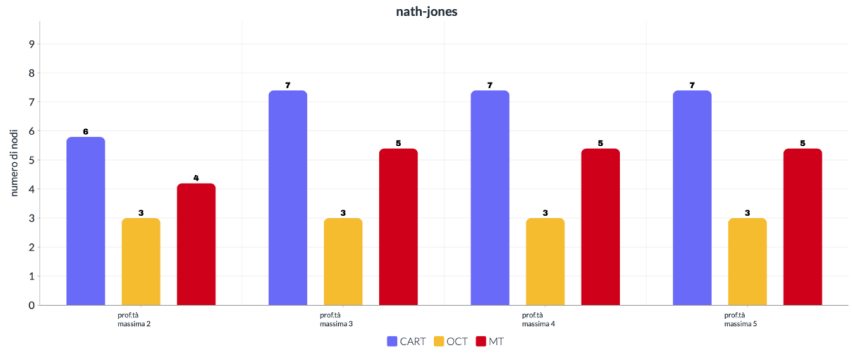
Nel contesto della costruzione di alberi decisionali, è cruciale considerare la complessità degli alberi generati da diversi algoritmi. In questa sezione, si esamina la complessità di alberi decisionali indotti da tre algoritmi CART, OCT e MT negli esperimenti svolti sugli insiemi di dati 'nath-jones', 'breast-cancer' e 'tic-tac-toe', riassunti nella tabella 3.2 con i risultati dell'esperienza esposti nelle fig. 3.3.

L'algoritmo CART è noto per la sua efficacia nella costruzione di alberi decisionali. Tuttavia, la sua tendenza a generare alberi complessi può portare a problemi di sovraadattamento, specialmente se non viene applicata una potatura adeguata.

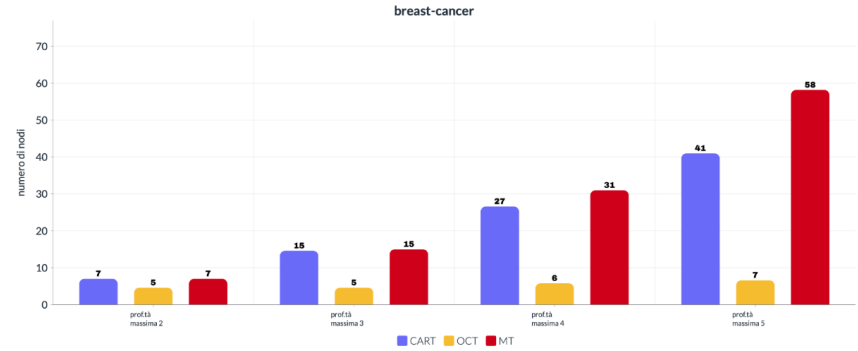
Al contrario, OCT si distingue per la sua capacità di costruire alberi decisionali ottimali. Questo processo di ottimizzazione porta a una maggiore precisione nella selezione delle caratteristiche e alla costruzione di alberi più snelli. La potatura applicata da OCT contribuisce a ridurre la complessità dell'albero, mitigando il rischio di sovraadattamento.

L'algoritmo MurTree invece, focalizzato sull'ottimizzazione globale, si propone di ridurre la complessità degli alberi decisionali mantenendo la massima precisione. La sua implementazione specifica, adatta a variabili binarie, contribuisce anch'essa a generare alberi meno complessi.

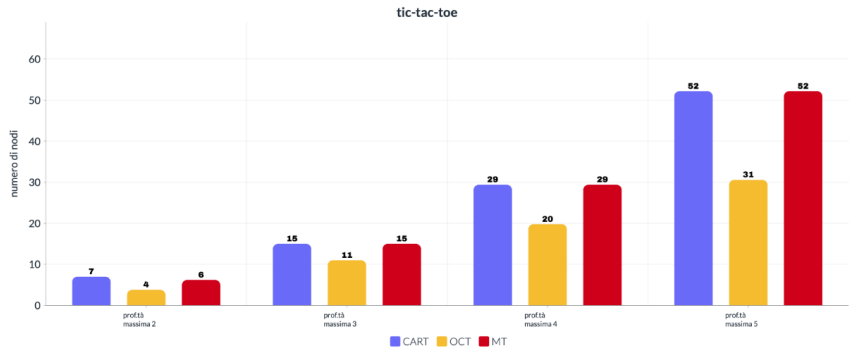
Attraverso un'analisi comparativa dei risultati ottenuti dagli algoritmi CART, OCT e MT su diversi dataset, emerge una tendenza chiara, ovvero, OCT grazie alla sua strategia di ottimizzazione e potatura, induce alberi decisionali con una complessità significativamente inferiore rispetto a quelli generati da CART. Invece, MT non sempre elabora DTs più semplici rispetto a CART, generalmente questo avviene quando la profondità massima è superiore a 4, come si vede dalle fig. 3.3b.



(a) Esperimento condotto sull'insieme di dati 'nath-jones'



(b) Esperimento condotto sull'insieme di dati 'breast-cancer'



(c) Esperimento condotto sull'insieme di dati 'tic-tac-toe'

Figura 3.3: Grafici mostranti il numero di nodi degli alberi decisionali indotti da CART, OCT e MT su tre insiemi diversi

Questa riduzione della complessità si traduce in una maggiore interpretabilità e generalizzazione, mitigando gli effetti del sovraadattamento, come si può notare dalle fig. 3.4, fig. 3.5 e fig. 3.6, che rappresentato la dispersione sull'accuratezza di probabilità di previsione dei modelli predittivi in funzione della complessità del DT. Si evince che alla maggior accuratezza corrisponde il minor numero di nodi.

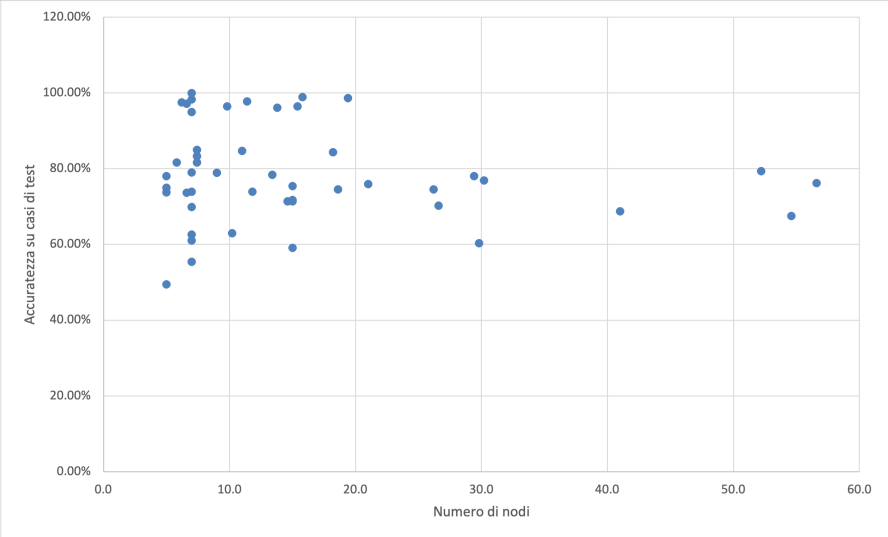


Figura 3.4: Dispersione tra Accuratezza e Complessità del DT del metodo CART.

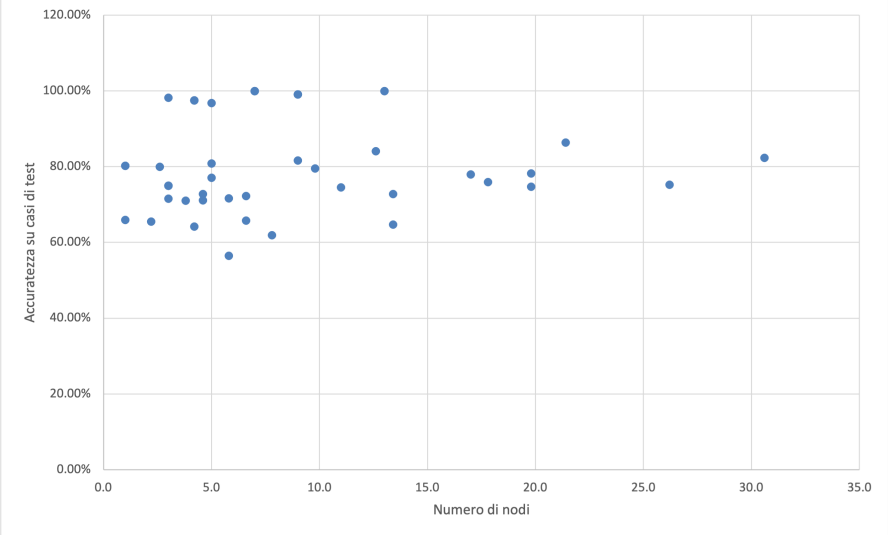


Figura 3.5: Dispersione tra Accuratezza e Complessità del DT del metedo OCT.

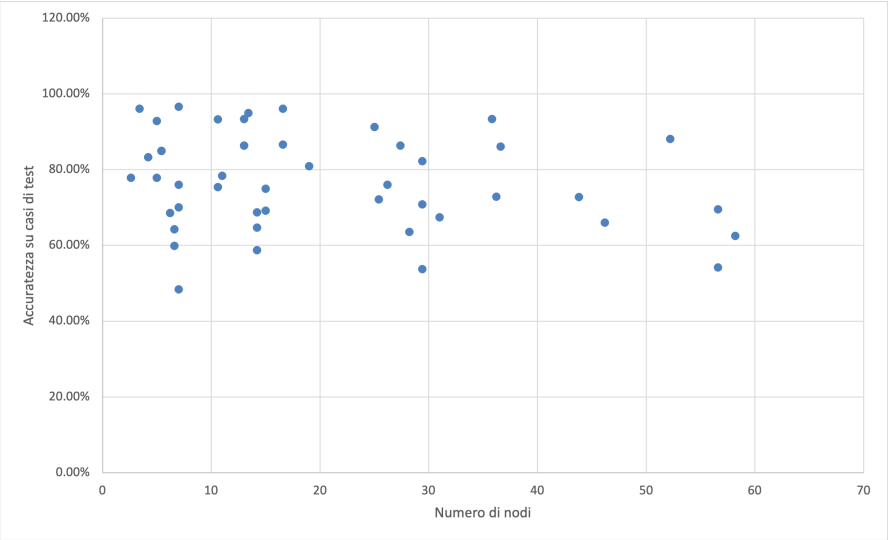


Figura 3.6: Dispersione tra Accuratezza e Complessità del DT del metodo MT.

3.4 Analisi dei Tempi di Induzione dei DTs

Dall'analisi dei tempi di induzione degli algoritmi di alberi decisionali (DTs), emerge una chiara differenza nelle prestazioni temporali tra gli algoritmi esaminati. Questi risultati sono stati riassunti nella tabella 3.4.

L'algoritmo CART, noto per la sua natura euristica, ha dimostrato di essere estremamente veloce nella creazione di DTs. In generale, i tempi di esecuzione di CART sono risultati inferiori ai centesimi di secondo per la maggior parte dei casi esaminati. Tuttavia, va notato che, nel caso specifico del dataset 'breast-cancer' con una profondità massima dell'albero pari a 5, il tempo di esecuzione è stato pari a un centesimo di secondo.

Allo stesso modo, l'algoritmo MT ha mostrato tempi di esecuzione quasi costantemente al di sotto del centesimo di secondo. Solo in alcuni casi, specialmente con una profondità massima dell'albero pari a 5, si sono registrati tempi leggermente superiori, raggiungendo qualche decimo di secondo. Il peggiore tempo di esecuzio-

ne per MT è stato di 0.43 secondi, osservato nel caso del dataset 'tic-tac-toe' con una profondità massima dell'albero pari a 5.

In contrasto, l'algoritmo OCT ha dimostrato una significativa lentezza nell'elaborazione dei DTs. I tempi di esecuzione per OCT sono stati costantemente registrati in decimi di secondo, con alcuni casi che hanno raggiunto anche l'ordine di mezzo secondo. Il tempo di esecuzione peggiore per OCT è stato di 2.91 secondi, riscontrato nel caso del dataset 'tic-tac-toe' con una profondità massima dell'albero pari a 5.

L'analisi dei tempi di induzione dei DTs evidenzia chiaramente le differenze sostanziali nelle prestazioni temporali tra gli algoritmi CART, OCT e MT, con CART e MT che operano quasi sempre in tempi estremamente rapidi, mentre OCT si distingue per tempi di esecuzione più prolungati.

Tabella 3.4: Tempo di addestramento in secondi - Parte 1

Insieme di Dati	prof.tà massima	CART	OCT	MT
nath-jones	2	0.00	0.18	0.00
nath-jones	3	0.00	0.20	0.00
nath-jones	4	0.00	0.25	0.00
nath-jones	5	0.00	0.29	0.00
soybean-small	2	0.00	0.20	0.00
soybean-small	3	0.00	0.25	0.00
soybean-small	4	0.00	0.31	0.00
soybean-small	5	0.00	0.37	0.00
monks-3	2	0.00	0.32	0.00
monks-3	3	0.00	0.68	0.00
monks-3	4	0.00	0.74	0.00
monks-3	5	0.00	0.84	0.00
monks-1	2	0.00	0.27	0.00
monks-1	3	0.00	0.48	0.00
monks-1	4	0.00	0.78	0.00
monks-1	5	0.00	1.07	0.00
hayes-roth	2	0.00	0.19	0.00
hayes-roth	3	0.00	0.40	0.00
hayes-roth	4	0.00	0.52	0.00
hayes-roth	5	0.00	0.56	0.20
monks-2	2	0.00	0.26	0.00
monks-2	3	0.00	0.51	0.00
monks-2	4	0.00	0.76	0.00
monks-2	5	0.00	1.26	0.00

Tabella 3.4: Tempo di addestramento in secondi - Parte 2

Insieme di Dati	prof.tà massima	CART	OCT	MT
house-votes-84	2	0.00	0.21	0.00
house-votes-84	3	0.00	0.33	0.00
house-votes-84	4	0.00	0.43	0.00
house-votes-84	5	0.00	0.47	0.00
spect	2	0.00	0.26	0.00
spect	3	0.00	0.32	0.00
spect	4	0.00	0.58	0.00
spect	5	0.00	0.69	0.20
breast-cancer	2	0.00	0.76	0.00
breast-cancer	3	0.00	1.56	0.00
breast-cancer	4	0.00	1.75	0.21
breast-cancer	5	0.01	1.84	0.24
balance-scale	2	0.00	0.70	0.00
balance-scale	3	0.00	0.75	0.00
balance-scale	4	0.00	1.59	0.00
balance-scale	5	0.00	1.99	0.20
tic-tac-toe	2	0.00	0.47	0.00
tic-tac-toe	3	0.00	1.00	0.00
tic-tac-toe	4	0.00	1.86	0.00
tic-tac-toe	5	0.00	2.91	0.43
car-evaluation	2	0.00	0.56	0.00
car-evaluation	3	0.00	1.12	0.00
car-evaluation	4	0.00	1.54	0.00
car-evaluation	5	0.00	1.76	0.21

Capitolo 4

Conclusioni

Per concludere, l'obiettivo di questo elaborato era valutare e confrontare le performance di tre algoritmi di induzione di alberi decisionali: Classification and Regression Trees (CART), Optimal Classification Trees (OCT), e MurTree (MT), nell'ottica di determinare la loro precisione nella previsione di esempi di test e complessità di DT indotto.

Dai risultati ottenuti emerge chiaramente che gli algoritmi ottimali (OCT e MT) tendono a generare alberi decisionali con una maggiore accuratezza rispetto agli algoritmi euristici come CART. Tale risultato potrebbe essere associato alla propensione degli algoritmi euristici a sovradattare i modelli, il che li rende meno precisi nella classificazione di dati non visti.

Inoltre, l'analisi della complessità degli alberi generati ha evidenziato come OCT sia in grado di produrre modelli con una complessità inferiore rispetto a CART, grazie alla sua strategia di ottimizzazione e potatura. Tuttavia, l'algoritmo MT non sempre genera alberi meno complessi rispetto a CART, dimostrandosi generalmente più efficiente con profondità massime minori o uguali a 4.

Riguardo ai tempi di induzione, si è osservato che CART e MT tendono ad avere tempi di esecuzione molto brevi, spesso nell'ordine dei centesimi di secondo, mentre

OCT risulta significativamente più lento, con tempi di esecuzione costantemente registrati in decimi di secondo, talvolta raggiungendo anche il secondo.

In conclusione, l'analisi comparativa ha evidenziato che, sebbene CART sia noto per la sua rapidità e semplicità, OCT risulta vantaggioso per la precisione e la complessità ridotta dei modelli, mentre MT offre un compromesso accettabile tra accuratezza e tempi di esecuzione, risultando particolarmente efficace su dataset di grandi dimensioni.

Infine, per ulteriori sviluppi futuri, si potrebbe approfondire l'analisi su altri tipi di dataset, valutare l'efficacia degli algoritmi in contesti specifici e investigare strategie per migliorare le prestazioni di CART e OCT senza compromettere la loro accuratezza.

Bibliografia

[1] Leo Breiman, Jerome Friedman, and Charles J Stone. *Classification and regression trees*. CRC press, 1984.

[2] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[3] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

[4] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, April 2017.

[5] Sicco Verwer and Yingqian Zhang. Learning decision trees with flexible constraints and objectives using integer optimization. In *Integration of AI and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings 14*, pages 94–103. Springer, 2017.

[6] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1625–1632, 2019.

[7] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1418–1426, July 2019.

[8] Pranav Ashok, Mathias Jackermeier, Pushpak Jagtap, Jan Křetínskỳ, Maximilian Weininger, and Majid Zamani. dtcontrol: Decision tree learning algorithms for controller representation. In *Proceedings of the 23rd international conference on hybrid systems: Computation and control*, pages 1–7, 2020.

[9] Emir DemiroviÄ‡, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23(26):1–47, 2022.

[10] Detlof Von Winterfeldt and Ward Edwards. *Decision Analysis and Behavioral Research*. Cambridge University Press, Cambridge [Cambridgeshire] ; New York, 1st edition, 1986. BF441 .V66 1986.

[11] Bogumił Kamiński, Michał Jakubczyk, and Przemysław Szufel. A framework for sensitivity analysis of decision trees. *Central European journal of operations research*, 26:135–159, 2018.

[12] LLC Interpretable AI. Interpretable ai documentation, 2023.

[13] Michael Hahsler and Ian Johnson. *arulesCBA: Classification Based on Association Rules*, 2023. R package version 1.2.6.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn:

Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Niels Lohmann. JSON for Modern C++, August 2022.

[16] Farzeen Zehra, Maha Javed, Darakhshan Khan, and Maria Pasha. Comparative analysis of c++ and python in terms of memory and time. *Preprints*, December 2020.