UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN "INGEGNERIA INFORMATICA"

Algoritmi di Interpretable Machine Learning con applicazioni alla bioinformatica

Relatore: *Professor Leonardo Pellegrina*

Laureando: *Riccardo Modolo (2009667)*

Correlatore: *Professor Fabio Vandin*

Anno Accademico: *2022/2023*

Data di Laurea: *25/09/2023*

Abstract

I modelli d'*Interpretable Machine Learning* sono essenziali per aumentare l'adozione del Machine Learning all'interno della società. Uno di questi viene chiamato **decision rules**, e fornisce una serie di regole atte a descrivere il criterio decisionale utilizzato dal modello. CORELS si occupa di fornire l'insieme di regole *ottimo* dato un modello. In questo documento verrà utilizzato CORELS in modo da fornire delle regole che distinguano due tipologie di tumore in base alle mutazioni genetiche dei pazienti. Nel corrente contesto dunque, il modello delle decision rules verrà utilizzato per osservare relazioni tra mutazioni somatiche e la tipologia tumorale.

Indice

Introduzione

Alcuni modelli di Machine Learning (ML) vengono definiti come **black box ML**. La black box è un modello che descrive un processo tramite il solo input e output. L'efficacia di tale costrutto risiede nella descrizione di sistemi complessi, dove l'importante è definire ad alto livello, quali processi vengono attuati, senza approfondirne il funzionamento intrinseco.

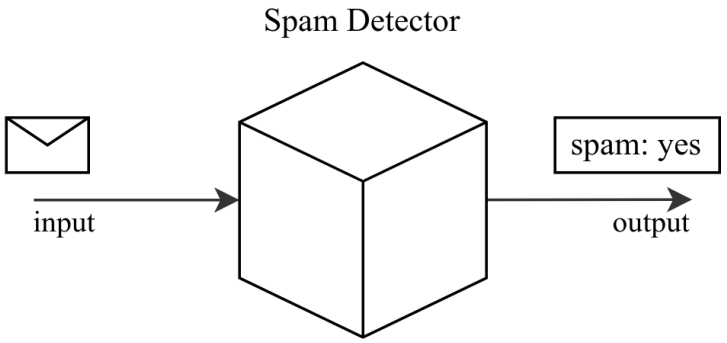


Figure 1: Esempio di black box: un modello che decide se una mail deve andare nella casella *spam*

La black box, per definizione, non si presta a una forte trasparenza del modello che rappresenta. Nel contesto ML, questo implica che il modello impara da solo attraverso i dati forniti, senza controllo di agenti esterni. Nonostante in alcuni contesti, questa mancanza di trasparenza possa risultare ininfluente o addirittura vantaggiosa, se si vuole in futuro fare largo uso di modelli ML, anche in ambienti sociali o sanitari, occorre avere una profonda conoscenza di essi.

Per aumentare la trasparenza, sono stati introdotti diversi modelli, detti *interpretable models*, i quali hanno il compito di spiegare la ragione dietro una predizione del modello ML. Uno di questi è il **decision rules**, il quale offre diverse condizioni *IF-THEN* per spiegare quali scelte portano a una particolare predizione.

CORELS è un algoritmo che punta a fornire il set di regole **ottimo**, ovvero con la maggiore precisione e il minor numero di regole, atto a interpretare il modello analizzato.

Le regole ottenute attraverso CORELS sono un funzionale elemento diagnostico, infatti, in questo modo, è facilmente visibile la presenza di *bias* o comportamenti discriminatori da parte del modello (in principio è stata questa la ragione principale che ha portato allo sviluppo di CORELS), come pure le dinamiche di **causa - effetto**.

La potenza delle **decision rules** risiede proprio nella semplicità con cui persone non esperte possano interpretare e valutare il modello di ML. Grazie agli *interpretable models* si riesce dunque a utilizzare molto più largamente il Machine Learning come strumento di supporto in diversi settori.

Chapter 1

Introduzione ai Tumori

Secondo AIOM (Associazione Italiana di Oncologia Medica), nel documento *"i numeri del cancro in Italia 2022"*, erano state stimate 390.700 nuove diagnosi di tumori. Nel primo semestre del 2020 il numero di decessi causati da tumori ammontavano a 48.449 per gli uomini e 39.290 per le donne. Queste cifre fanno riflettere in merito a quanto questa *malattia dei geni* sia letale e diffusa. Il tumore nasce attraverso diverse mutazioni dei geni nel tempo, che portano ad anomalie funzionali, per questa ragione lo studio dell'assetto molecolare dei tumori è il campo di ricerca più promettente alla cura di esso.

1.1 Geni

I geni sono l'unità ereditaria fondamentale degli organismi viventi, sono composti da una sequenza di nucleotidi, le unità costitutive del DNA (A, C, G, T). Ogni gene contiene informazioni atte allo sviluppo fisico di un essere vivente, per esempio molti geni si occupano della codifica di proteine, macromolecole essenziali nei processi interni delle cellule. L'alterazione di geni è responsabile di alcune malattie di natura ereditaria, e anche di predisposizioni ad altre malattie.

1.2 Mutazioni

Le mutazioni genetiche avvengono spontaneamente all'interno del nostro organismo, alcune vengono riparate naturalmente, altre passano inosservate, portando al potenziale sviluppo di un tumore. Quando si parla di mutazioni, si intende l'alterazione del normale funzionamento di alcune cellule, causato dall'inquinamento delle informazioni genetiche, alcune di esse potrebbero essere:

1. **Presenza dei geni oncogeni**: i geni oncogeni sono geni alterati che favoriscono lo sviluppo tumorale. Nella loro forma normale (geni proto-oncogeni) hanno il compito di regolare le proprietà replicative delle cellule (per esempio la riparazione di tessuti). In caso di alterazione, le informazioni genetiche potrebbero imporre alla cellula un processo replicativo a tasso elevato, alterando l'equilibrio del sistema.

2. **Alterazione dell'apoptosi**: l'apoptosi è una "morte programmata" delle cellule. Questa caratteristica è essenziale per contenere il numero di cellule all'interno di un sistema.

3. **Modifica ai geni di soppressione tumorale**: esistono dei geni chiamati **oncosoppressori** che si occupano di bloccare progressioni cellulari indesiderate. Una alterazione di essi potrebbe alterare le capacità d'individuare cellule alterate e dare via libera alla proliferazione di tratti tumorali.

Per scoprire la presenza di mutazioni si può optare per dei test genetici, previa consultazione di un medico genetista. I tumori posso essere classificati in:

1. **tumori benigni**: le mutazioni tendono alla proliferazione delle cellule tumorali ma solo nel contesto locale di creazione

2. **tumori maligni**: in questo caso le mutazioni viaggiano e si espandono all'interno del corpo, comportando una alterazione totale del sistema

1.2.1 Cause delle mutazioni

Le mutazioni si possono suddividere in due tipologie: **costitutive** e **acquisite**. Le seconde identificano quelle mutazioni non ereditate dai genitori, generalmente le cause di queste mutazioni sono:

1. **fattori ambientali**:
   (a) inquinamento atmosferico (esposizione alle polveri fini, le quali sono causa di un aumento della mortalità per cause naturali in generale)
   (b) agenti chimici (i più famosi sono il *benzene* e l'*amianto*)
   (c) agenti fisici (l'esposizione ai raggi X, che può aumentare il rischio di tumori)
   (d) agenti infettivi (sebbene il cancro non sia una malattia infettiva, alcune infezioni come l'**HIV** e il **Papilloma virus umano** favorirebbero la proliferazione di cellule tumorali)
2. **stile di vita**:
   (a) fumo di tabacco (causa di circa un terzo delle morti di cancro secondo l'*Istituto Superiore di Sanità*)
   (b) esposizione eccessiva al sole e raggi ultravioletti (soprattutto nei bambini)
   (c) consumo eccessivo di alcol (anche un consumo moderato aumenta le probabilità di tumori)
   (d) alimentazione sbilanciata (in particolare se eccede il consumo di carne rossa e insaccati)
   (e) sedentarietà e obesità

Per quanto riguarda le mutazioni **costitutive**, ovvero quelle presenti fin dalla nascita, aumentano il rischio rispetto ai non portatori della mutazione, ma non comportano una inevitabile comparsa del cancro. C'è da fare presente che, in una visione completa di "componente familiare", nel campo della familiarità del cancro, non sussiste solo la trasmissione delle componenti genetiche, ma anche abitudini e stili di vita.

1.2.2 Single Nucleotide Variant

Quando avviene una mutazione del materiale genetico a carico di un singolo nucleotide si parla di **Single Nucleotide Polymorphism (SNP)**. Viene chiamato invece **Single Nucleotide Variant (SNV)**, quando è presente una **SNP** osservabile su una percentuale della popolazione sotto l'1%.

Presa una sequenza di DNA da due pazienti:

ATCCT**T**AC e ATCCT**C**AC,

si può dire che nella frequenza è presente un SNP che differenzia i due alleli C e T. Lo studio degli **SNV** è importante come elemento diagnostico, ci permette di capire come una alterazione genetica porti a una patologia. Essendo il cancro una malattia dei geni, la difficoltà principale nella sua cura sta nella *sartorialità* della malattia stessa, ovvero dipende dalle alterazioni genetiche del singolo. Nonostante ciò, cercare delle mutazioni comuni nello sviluppo del cancro potrebbe essere una scelta vincente per quando riguarda la prevenzioni o per evidenziare dove concentrare gli sforzi nello studio delle cure.

Chapter 2

Interpretable Machine Learning

2.1 Interpretazione

Se si vuole discutere di rendere un modello *interpretabile*, occorre offrire una definizione del termine:

"interpretability is the **degree** to which a human can understand the cause of a decision." [13, O. Biran, C. Cotton]

L'interpretazione dunque, non offre una spiegazione, ma mostra risultati su cui l'utente finale dovrà effettuare un lavoro di comprensione. Questo concetto poi, è fortemente correlato a quello di causalità, ovvero tentare di spiegare e modellare le cause di un evento. Sulla causalità è stato fatto un lavoro considerevole nel contesto dell'IA, uno dei nomi più noti è Judea Pearl[3], famoso per l'invenzione di un modello causale che presenta variabili *esogene* ed *endogene*.

Una nota finale importante, risulta quella della distinzione tra *Causal Attribution* e *Causal Explanation*: poter avere a disposizione le relazioni di causa tra gli eventi (*attribution*), non implica necessariamente fornire una spiegazione (*explanation*) di essi.

2.2 Alcuni modelli noti

I modelli d'Interpretable Machine Learning (IML) vengono ottenuti attraverso determinati *algoritmi*. In questa sezione verranno accennati alcuni modelli alternativi a quello delle **decision rules**, il quale verrà largamente spiegato per tutto il resto del documento.

2.2.1 Linear Regression

Linear Regression è un modello che si basa sulla somma pesata delle features del dataset. Presa y una label e $\mathbf{x}$ il vettore di features, la relazione label-features viene formalizzata come segue:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \epsilon$$

Epsilon (ϵ) è il parametro di errore commesso, mentre il primo valore della funzione (β_0), il quale non moltiplica nessuna features, viene chiamato *intercept* e ci indica il punto d'intersezione con l'asse y.

Lo scopo del modello è cercare la composizione di pesi ($\hat{\boldsymbol{\beta}}$) ottimale per predire il maggior numero possibile di label. Per calcolare quanto preciso sia il vettore dei pesi, uno dei metodi consiste nel calcolo del *Sum Squared Error (SSE)*, ovvero il quadrato della differenza tra la label effettiva e il valore ottenuto attraverso la funzione.

$$\hat{\boldsymbol{\beta}} = \underset{\beta_0,\ldots,\beta_p}{\operatorname{argmin}} \sum_{i=0}^{N} \left(y - \left(\beta_0 + \sum_{j=1}^{p} \beta_j \cdot x_j \right) \right)^2$$

Questo modello evidenzia bene quanto un parametro influenzi il valore finale rispetto ad altri. Uno degli svantaggi principali però, è la sua linearità: gli ambienti completamente lineari, nel quotidiano, sono un gruppo ristretto e tendono a semplificare la realtà. In secondo luogo, è difficile interpretare i valori di una linear regression, in quanto alcuni pesi potrebbero risultare, agli occhi dell'utente finale, controintuitivi.

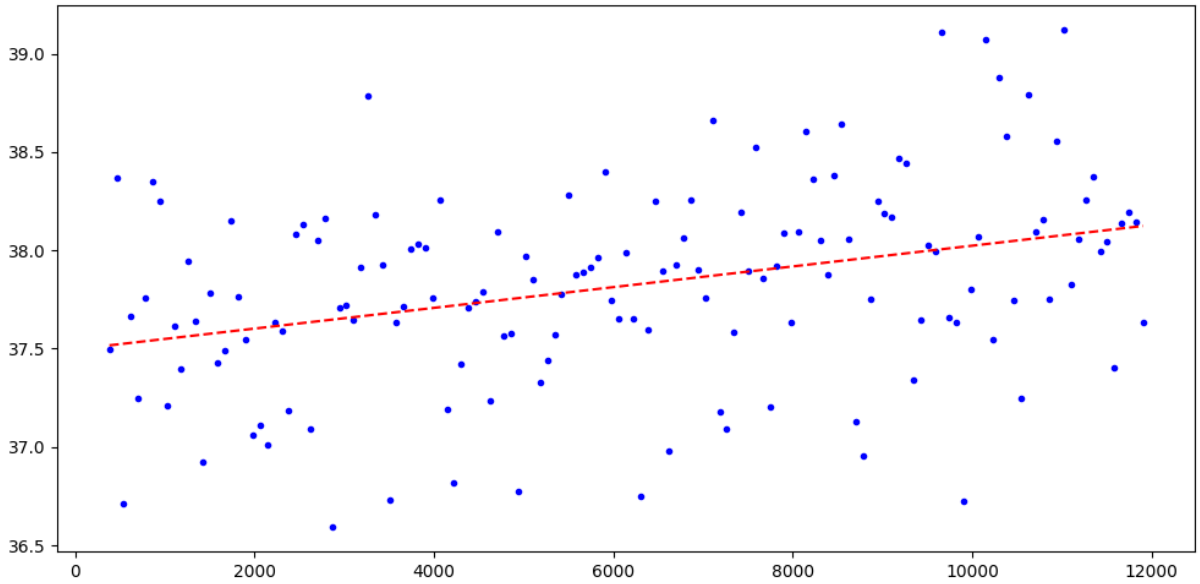


Figure 2.1: Grafico del processo di linearizzazione di un set di dati

2.2.2 Decision Trees

Il primo punto di forza dei decision trees è la loro efficacia in sistemi **non lineari**, questo aumenta le opportunità di utilizzo di questo modello. La struttura ad albero comprende i *nodi interni*, i quali corrispondono alle diverse features presenti nel dataset, le *foglie*, le quali offrono il risultato della predizione, e i *rami*, i possibili valori (o range di valori) assumibili dalla feature.

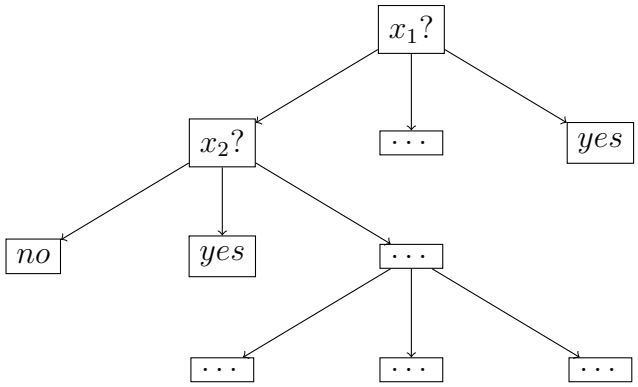


Figure 2.2: Esempio di decision tree, ogni ramo corrisponde a un possibile valore della feature, ogni foglia corrisponde a una predizione della label

L'algoritmo più famoso per la generazione di un *decision tree* è CART (Classification And Regression Tree algorithm). Ad alto livello, basti pensare che, come l'intuizione potrebbe portare a credere, l'ordine dei nodi è significativo e viene calcolata l'*importanza* di ogni possibile nodo, per decidere quale inserire nell'albero. Per il calcolo dell'importanza viene utilizzato il concetto di **entropia**, tipico della teoria dell'informazione.

Questo modello è molto vicino a quello che è il *modus operandi* umano nel processo di decisione, perciò è facilmente interpretabile: offre visivamente le interazioni tra le features, le relazioni di causa effetto e cosa si deve alterare per manipolare il risultato finale. Si può affermare dunque che questo tipo di modello è *"Human-Friendly"*[14, section 3.6]

Chapter 3

Componenti di CORELS

Verranno analizzate e formalizzate, in questa sezione, tutte le componenti fondamentali al funzionamento di CORELS e alle dimostrazioni dei lemmi presenti in questo documento.

3.1 Dataset

Un dataset consiste in un insieme di oggetti generici su cui si desidera effettuare un apprendimento, questo insieme verrà chiamato $\mathbf{X}$. Un oggetto appartenente a questo insieme, viene rappresentato come un vettore delle sue *features*, ovvero gli attributi che caratterizzano l'oggetto.

$$\mathbf{x_n} \in \mathbf{X} \qquad \mathbf{x_n} = \{x_0, x_1, \ldots, x_K\} \tag{3.1}$$

Per le caratteristiche sopra elencate, i dataset si prestano bene a una rappresentazione tabellare, dove ogni riga rappresenta un singolo oggetto e le colonne le *features* di esso.

sex	age	studytime	Dalc	Walc	Absences	Grade 1
F	18	2	1	1	6	**5**
F	17	2	1	1	4	**5**
F	16	2	1	2	4	**6**
M	16	2	1	2	10	**15**
M	16	2	1	1	0	**12**

Table 3.1: Una sezione filtrata di un dataset che vuole studiare l'influenza del consumo di alcol sulle performance scolastiche [4]

Come si può notare dall'esempio, non esiste limitazione ai possibili valori che possono assumere le *features* di un oggetto. L'ultima colonna viene chiamata **label**, essa rappresenta il valore che si cerca di predire.

CORELS opera con **dataset binari**, i quali vincolano i valori di ogni feature ad assumere valori booleani $x_j \in \{0, 1\}$ (questa condizione vale anche per la label). Sebbene questa *"limitazione"* possa sembrare molto restrittiva, non rappresenta un problema. È possibile, infatti, trasformare dataset non binari in binari, tramite opportune condizioni sui valori che può assumere quella caratteristica.

L'insieme delle tuple $\{(x_n, y_n)\}_{n=1}^{N}$ presenti nel file, sono noti come **training data**, e proprio su questi oggetti verranno create le regole.

3.2 Rule

Una regola è una condizione *IF-THEN* che fornisce informazioni sul valore della label rispetto a determinate condizioni. Una regola associa a una condizione p, una determinata predizione $q \in \{0, 1\}$

$$r_k = p_k \rightarrow q_k \tag{3.2}$$

Prendiamo come esempio le regole trovate da CORELS per risolvere il problema *"il carcerato commetterà reati entro due anni dalla sua scarcerazione?"* [8].

if $(age = 18 - 20)$ **and** $(sex = male)$ **then predict** yes	**if** p_1 **then predict** q_1
else if $(age = 21 - 23)$ **and** $(priors = 2 - 3)$ **then predict** yes	**else if** p_2 **then predict** q_2
else if $(priors > 3)$ **then predict** yes	**else if** p_3 **then predict** q_3
else predict no	**else predict** q_0

Ogni riga corrisponde a una regola. Una serie di regole che si susseguono vengono chiamate *regole associative*, dove, se la condizione della attuale regola non viene rispettata, si passa alla regola successiva, fino alla **regola di default**, la quale potrebbe essere intesa come $r_0 : true \rightarrow q_0$.

3.3 Rule List

Un insieme di regole associative viene definito come **rule list**:

$$d = (r_1, r_2, \ldots, r_K, r_0) \qquad K > 0 \tag{3.3}$$

Consiste in $(K+1)$ tuple di regole distinte. Questo non è l'unico modo di rappresentare d, infatti, nelle successive dimostrazioni, verrà usata la seguente notazione:

$$d = (d_p, \delta_p, q_0, K). \tag{3.4}$$

Tale forma si ottiene separando r tra condizioni e predizioni, $d_p = (p_1, p_2, \ldots, p_K)$, $\delta_p = (q_1, q_2, \ldots, q_K)$, q_0 è la previsione della regola di default e K numero di regole, esclusa quella di default.

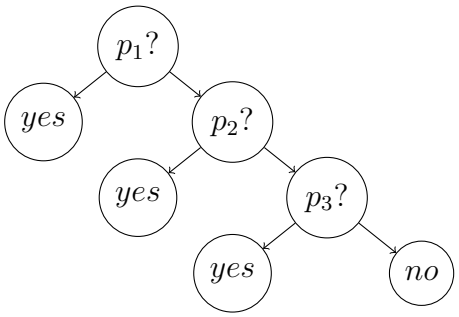


Figure 3.1: Albero binario di una serie di regole associative, una rule list

Occorre evidenziare che, seppur simili, *decision trees* e *rule lists* sono differenti: nel primo, la composizione dell'albero vede come nodo interno una sola feature ed espande l'albero in base alle features più rilevanti, le rule lists invece, hanno come nodo interno le **condizioni**. Le condizioni possono includere più features e più valori di esse in relazione tra loro. Sotto un certo aspetto, le rule lists sono una versione contratta dei decision trees.

3.3.1 Prefisso

Data una lista di condizioni $d_p = (p_1, \ldots, p_k, \ldots, p_K)$, per ogni $k \leq K$ possiamo definire una sottolista $d_p^k = (p_1, \ldots, p_k)$ che verrà chiamata **k-prefisso** di d_p. Verrà detto che d_p inizia con d_p^k. Si definisce anche $\sigma(d_p)$ come l'insieme di tutte le rule list che iniziano con d_p (quindi hanno come k-prefisso d_p):

$$\sigma(d_p) = \{(d'_p, \delta'_p, q'_0, K') : d'_p \text{ che inizia con } d_p\}. \tag{3.5}$$

Se d'_p estende d_p di una singola condizione, verrà detto che d'_p è figlio di d_p e, di conseguenza, d_p è padre di d'_p.

3.3.2 Funzione di Cattura

Un oggetto x_n viene detto *"catturato"* quando realizza le condizioni di almeno una regola della rule list d (viene esclusa la regola di default r_0). Essendo le regole della rule list di tipo associativo, la label prevista corrisponderà al risultato della prima regola che cattura x_n. Dunque, se la prima regola rispettata è differente dalla **regola di default** verrà detto che d_p *cattura* x_n, altrimenti d_p ***NON*** *cattura* x_n. Preso β come un insieme di condizioni:

$$cap(x_n, \beta) \qquad 1 \rightarrow \beta \text{ cattura } x_n, \qquad 0 \rightarrow \beta \textbf{ NON } \text{cattura } x_n \tag{3.6}$$

È importante notare che, presa qualsiasi lista di condizioni d_p, e un suo prefisso d'_p, ogni x_n catturato da d'_p viene catturato anche da d_p:

$$\{x_n : cap(x_n, d'_p)\} \subseteq \{x_n : cap(x_n, d_p)\}. \tag{3.7}$$

Presa una lista di condizioni d_p, viene preso un suo sottoinsieme β. Per capire se x_n viene catturato in β, ovvero una zona specifica della rule list, viene definita la nuova funzione $cap(x_n, \beta|d_p)$. La funzione assume valore 1 se la prima condizione che cattura x_n è presente in β e 0 altrimenti. Si può osservare che $cap(x_n, \beta|d_p) = 1$ se e solo se $cap(x_n, \beta) = 1$, mentre $cap(x_n, \beta|d_p) = 0$ sia se $cap(x_n, \beta) = 0$, sia $cap(x_n, \beta) = 1$. Se $cap(x_n, \beta) = 1$ ma $cap(x_n, \beta|d_p) = 0$ vorrà dire che esiste una condizione appartenente a d_p che cattura x_n prima che lo faccia β. La formula risulta:

$$cap(x_n, p_k|d_p) = \left(\bigwedge_{k'=1}^{K-1} \neg cap(x_n, p'_k) \right) \cdot cap(x_n, p_k) \tag{3.8}$$

3.3.3 Funzione di Supporto

Viene definita la *funzione di supporto normalizzato* come il tasso di oggetti catturati dalla lista di condizioni β:

$$supp(\beta, \mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} cap(x_n, \beta) \tag{3.9}$$

In modo analogo, si definisce $supp(\beta, \mathbf{x}|d_p)$ come il tasso di oggetti catturati dalla lista di condizioni β nel contesto d_p:

$$supp(\beta, \mathbf{x}|d_p) = \frac{1}{N} \sum_{n=1}^{N} cap(x_n, \beta|d_p) \tag{3.10}$$

3.4 Objective Function

CORELS implementa una misura qualitativa di tipo *minimize loss function*. La funzione obiettivo è quindi un indicatore di ottimalità della rule list d.

$$R(d, \mathbf{x}, \mathbf{y}) = l(d, \mathbf{x}, \mathbf{y}) + K\lambda \tag{3.11}$$

l rappresenta la *loss function*, in questo caso, viene usato il rapporto di label predetti in modo errato attraverso le regole di d. La seconda parte dell'equazione viene chiamata *regularization function*, la quale ha lo scopo di ridurre l'**overfitting**.

La condizione di **overfitting** avviene quando il modello performa in maniera molto buona con il training data fornito, ma con dati non analizzati (nuovi) potrebbe risultare inutilizzabile. Il numero di regole, è un indice utile a capire il rischio di overfitting: più regole corrispondono a una maggiore specializzazione su quel training data, a discapito di una buona generalizzazione del problema. Grazie alla regularization function, l'aggiunta di una regola, viene visto come un incremento λ di errore di valutazione della predizione.

λ è un valore empirico, il quale può essere arbitrariamente modificato in base alle necessità e contesti del modello.

3.5 Valutazione del Modello

Quando ci si occupa di modelli di *Supervised Machine Learning* occorre trovare un modo per valutare quanto efficacemente riescano a predire la label. La funzione che si occupa di ciò viene chiamata **score** e offre come risultato il tasso di precisione del modello. Come si calcola lo score? Innanzitutto occorre precisare che la valutazione del modello si basa su dati che non sono mai stati precedentemente analizzati nella fase di addestramento. Dato questa nozione solitamente viene preso tutto il training data e diviso in due parti: ***training set*** il quale verrà usato per addestrare il modello a predire la label e ***test set*** il quale si occupa invece di ottenere lo score del modello. Solitamente il dataset non viene suddiviso in modo simmetrico, ma si propende per un 80% al training set e un 20% al test set. Quindi, in prima fase si addestra il modello con il training set e in secondo luogo si controlla quanti valori del test set il modello predice correttamente, così si ottiene lo score.

Chapter 4

Framework

4.1 Branch-and-Bound Framework

Branch and Bound è una strategia risolutiva legata ai **problemi di ottimizzazione**. Lo scopo principale risulta dunque trovare la soluzione ottima, attraverso una funzione di costo. Nel caso di CORELS la funzione di costo è l'*objective function*, dunque la soluzione ottima avrà la funzione di costo più piccola (commette meno errori possibili).

Il Framework prende l'insieme delle soluzioni ammissibili e lo suddivide in un gruppo di sottoinsiemi, successivamente, viene calcolato il *bound* di ogni sottoinsieme e vengono comparati con il risultato ottimo salvato precedentemente: se il bound è maggiore della attuale soluzione ottima, vorrà dire che il sottoinsieme non conterrà la soluzione ottima, perciò verrà eliminato. Continuando il procedimento secondo l'ordine di espansione dei sottoinsiemi, dettato attraverso determinate policy, si arriva a scartare una grande quantità di soluzioni non ottime, fino alla soluzione del problema.

Questi procedimenti possono essere visualizzati attraverso un albero di ricerca, partendo dal nodo *root*, ovvero l'insieme di tutte le soluzioni ammissibili, andremo a espandere dei nodi che corrispondono ai sottoinsiemi. Viene calcolato il bound di ogni nodo e viene deciso quale nodo espandere e quale interrompere completamente. Alla fine di tutta l'esecuzione verrà trovata la soluzione ottima in una foglia dell'albero.

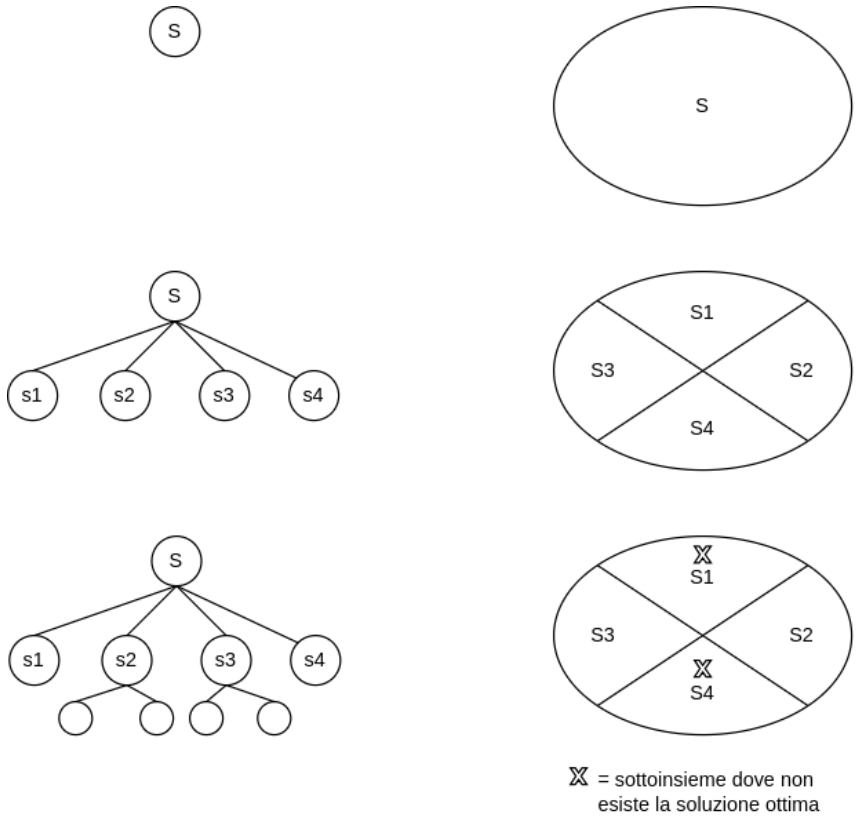


Figure 4.1: Visualizzazione dei procedimenti di un algoritmo Branch and Bound [10]

4.2 Panoramica delle Ottimizzazioni

Nella pratica, i ricercatori di CORELS hanno utilizzato questo framework, trovando utili bounds e osservazioni.

1. Il lower bound di un prefisso è utilizzabile da tutte le estensioni di esso
2. Se una rule list non è abbastanza accurata in relazione alla sua lunghezza, elimineremo tutte le sue estensioni
3. Possiamo ottenere *a priori* un limite superiore sulla lunghezza della rule list ottima
4. Una rule list ottima deve raccogliere un amplio numero dei dati inseriti, in questo modo possiamo garantire di ottenere le regole del modello analizzato
5. Il numero di osservazioni predette correttamente da ogni regola nella rule list ottima deve superare la funzione di *threshold*
6. Viene considerata solo la permutazione ottima di antecedenti in un prefisso, le altre possono essere omesse

4.3 Objective Lower Bound

Una delle dimostrazioni più importanti per i contenuti successi è il *lower bound* calcolato dall'algoritmo su una rule list. La caratteristica di maggiore interesse è l'ereditarietà di questo bound (caratteristica che rende il framework branch and bound funzionante), dunque, una rule list che ha un prefisso con un determinato bound lo può sfruttare a sua volta come bound. In questa sezione viene spiegato come si è ottenuto questo bound e la dimostrazione di ereditarietà di esso.

In primo luogo si scompone la funzione l della objective function in due parti: la prima gli errori di valutazione della rule list, la seconda gli errori di valutazione della regola di default:

$$l(d, \mathbf{x}, \mathbf{y}) = l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + l_0(d_p, q_0, \mathbf{x}, \mathbf{y}),$$

La prima parte della funzione risulta:

$$l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} cap(x_n, p_k | d_p) \wedge \mathbb{1}[q_k \neq y_n]$$

mentre la seconda parte:

$$l_0(d_p, q_0, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^{N} \neg cap(x_n, d_p) \wedge \mathbb{1}[q_0 \neq y_n]$$

La funzione $\mathbb{1}[condition]$, è una semplice funzione (molto frequente nella letteratura sul Machine Learning) che restituisce come risultato 1 quando la condizione è vera, 0 altrimenti. Se eliminano l'errore di default otteniamo un lower bound della funzione obiettivo:

$$b(d_p, \mathbf{x}, \mathbf{y}) = l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K \leq R(d, \mathbf{x}, \mathbf{y}) \tag{4.1}$$

Questo lower bound è stato concepito pensando che la prospettiva ottimale comprende il totale inutilizzo della regola di default, dato che essa non "interpreta" il modello, la regola di default deve essere vista dunque come il vincolo della rule list a fornire sempre una risposta. Quando non viene utilizzata in nessun contesto la regola di default verrà detto che la label è **pienamente determinata**. Il teorema proposto sotto (ripreso dal documento di CORELS[8]) dimostra che questo bound può essere ereditato da tutte le rule list che hanno come prefisso d.

Teorema 1. *Si prenda un lower bound $b(d_p, \mathbf{x}, \mathbf{y})$ come definito in (4.1). Viene utilizzato anche $\sigma(d_p)$ come l'insieme di tutte le rule list che hanno come prefisso d_p (3.5). Sia $d = (d_p, \delta_p, q_0, K)$ una rule list con prefisso d_p e $d' = (d'_p, \sigma'_p, q_0, K') \in \delta(d_p)$ una qualsiasi rule list con prefisso d'_p che inizia con d_p. Dato che $K' \geq K$, allora $b(d_p, \mathbf{x}, \mathbf{y}) \leq R(d', \mathbf{x}, \mathbf{y})$*

Proof. Per dimostrare ciò, si evidenzia come una qualsiasi estensione di $d_p = (p_1, \ldots, p_K)$, in questo caso $d'_p = (p_1, \ldots, p_K, \ldots, p_{K'})$ possa unicamente aggiungere errori:

$$
\begin{aligned}
l_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) &= \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K'} cap(x_n, p_k, |d'_p) \wedge \mathbb{1}[q_k \neq y_n] \\
&= \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{k=1}^{K} cap(x_n, p_k, |d_p) \wedge \mathbb{1}[q_k \neq y_n] + \sum_{k=K+1}^{K'} cap(x_n, p_k, |d'_p) \wedge \mathbb{1}[q_k \neq y_n] \right) \\
&= l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \frac{1}{N} \sum_{n=1}^{N} \sum_{k=K+1}^{K'} cap(x_n, p_k, |d'_p) \wedge \mathbb{1}[q_k \neq y_n] \geq l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) \qquad (4.2)
\end{aligned}
$$

Da questa disuguaglianza ricaviamo che:

$$
\begin{aligned}
b(d_p, \mathbf{x}, \mathbf{y}) &= l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K \\
&\leq l_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y}) + \lambda K' = b(d'_p, \mathbf{x}, \mathbf{y}) \leq R(d', \mathbf{x}, \mathbf{y}) \qquad (4.3)
\end{aligned}
$$

□

Dato che la soluzione ottima consiste nel minimizzare il numero di errori, grazie a questa dimostrazione, l'algoritmo può escludere dalla soluzione ottima tutte quelle rule list (e le estensioni di esse) che hanno il lower bound maggiore della Objective function ottima corrente R^c. R^c è un valore che decresce durante l'esecuzione dell'algoritmo, in questo modo CORELS riesce ad applicare la fase di bounding del paradigma *B&B*.

Chapter 5

Implementazioni

In questa sezione verrà fatto riferimento al codice di CORELS in C++[1]. Verranno analizzate ad alto livello le strutture dati principali della libreria e le policy di ricerca personalizzate.

5.1 Prefix Tree

Il **prefix tree**, comunemente conosciuto come *Trie*, è la struttura dati principale di CORELS. Nell'utilizzo per il quale il Trie è stato concepito, all'interno dell'albero vi vengono inserite stringhe, ove ogni nodo rappresenta un singolo carattere e i rami il collegamento tra le lettere. Ciò che rende unico un nodo all'interno della struttura dati è la sua posizione all'interno dell'albero, perciò possono essere presenti più elementi con lo stesso valore.

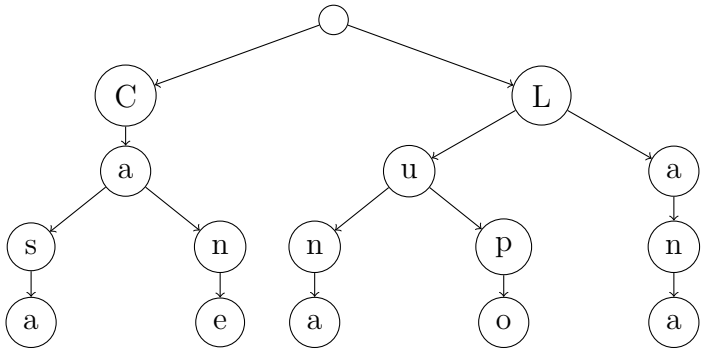


Figure 5.1: Esempio di Trie per la raccolta di parole italiane

Lo scopo di CORELS è di ottenere la **rule list ottima** per rappresentare il modello fornito. L'insieme di tutte le regole possibili verrà implementata con un Trie: ogni nodo corrisponderà a una singola regola, e un cammino rappresenterà una rule list.

Il vantaggio implementativo di CORELS di una struttura ad albero sta anche nel trasportare i calcoli dei bounds precedenti: CORELS calcola bound avvalendosi di quelli calcolati dai prefissi, secondo la proprietà di ereditarietà del bound dimostrata nel capitolo precedente.

5.2 Search Policies

CORELS implementa diversi algoritmi di ricerca all'interno del proprio codice (per utilizzare la terminologia del branch and bound: policy di branching), in modo tale da poter scegliere l'ordine di esplorazione del Trie. Le policy di ricerca implementate sono:

- Breath-first search (BFS)
- Depth-first search (DFS)
- Best-first search con la objective function (objective)
- Best-first search con l'errore di predizione minore (lower bound)
- Best-first search con la curious policy (curious)

Tutte le policy di ricerca vengono implementate attraverso una *priority queue*, contente le foglie del Trie. Grazie al principio di polimorfismo applicabile in c++, CORELS implementa le funzioni di comparazione attraverso veri e propri oggetti function (std::function).

5.3 Curiosity policy

La curious policy, è una strategia di ricerca personalizzata che sfrutta la **curiosity function**, una funzione che ha come obiettivo stimare quanto è conveniente espandersi su una determinata rule list.

Si pensi alla curiosità come al valore atteso della objective function di una rule list d', figlia di quella attuale d, in questo modo è intuitivo pensare che l'algoritmo privilegi valori attesi bassi, in quanto la prospettiva di sbagliare meno "incuriosisce" maggiormente l'algoritmo.

$$C(d_p, \mathbf{x}, \mathbf{y}) = \mathbb{E}[l(d'_p, \delta'_p, \mathbf{x}, \mathbf{y})] + \lambda \mathbb{E}[K'] \tag{5.1}$$

Quando si decide d'implementare una policy di ricerca personalizzata, basata su una nuova funzione da calcolare, occorre effettuare delle valutazioni in rapporto *costi - benefici*: da una parte una potenziale riduzione dell'espansione dei nodi all'interno dell'albero, dall'altra il costo computazionale nel calcolare questa funzione. Nel codice finale viene implementato un tipo di curiosity, calcolabile attraverso il lower bound (4.1) e la funzione di supporto (3.9), parametri già precedentemente calcolati e presenti all'interno del nodo in analisi, questo comporta un notevole vantaggio computazionale.

Per capire come viene implementata la curiosity, verrà percorso passo a passo, come si ottiene la formula finale. Vengono definite due rule list:

$$d = (d_p, \lambda_p, q_0, K) \qquad d' = (d'_p, \lambda'_p, q'_0, K')$$

dove d_p è prefisso di d'_p. Si definisce N_{cap} come il numero di oggetti catturati da d_p.

$$N_{cap} = \sum_{n=1}^{N} cap(x_n, d_p)$$

$$\mathbb{E}[K'] = \frac{N}{N_{cap}} \cdot K \tag{5.2}$$

Per la seconda parte della formula viene specificato che un antecedente qualsiasi d'_p commetta gli stessi errori del suo prefisso d_p, l'errore di classificazione atteso risulta

$$\begin{aligned} \mathbb{E}[l(d_p, \delta_p, \mathbf{x}, \mathbf{y})] =& \mathbb{E}[l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y})] + \mathbb{E}[l_0(d_p, q_0, \mathbf{x}, \mathbf{y})] \\ & \mathbb{E}[l_p(d'_p, \delta'_p, \mathbf{x}, \mathbf{y})] = \mathbb{E}[K']\Big(\frac{l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y})}{K}\Big) \end{aligned} \tag{5.3}$$

$l_0(d_p, q_0, \mathbf{x}, \mathbf{y}) = 0$ in quanto si assume non venga usata in alcun modo la regola di default. Ora si può riscrivere (5.1) con le equazioni esplicitate sopra:

$$
\begin{aligned}
C(d_p, \mathbf{x}, \mathbf{y}) &= \left(\frac{N}{N_{cap}}\right)\Big(l_p(d_p, \delta_p, \mathbf{x}, \mathbf{y}) + \lambda K\Big) \\
&= \left(\frac{1}{N}\sum_{n=1}^{N} cap(x_n, d_p)\right)^{-1} \cdot b(d_p, \mathbf{x}, \mathbf{y}) = \frac{b(d_p, \mathbf{x}, \mathbf{y})}{supp(d_p, \mathbf{x})}
\end{aligned} \tag{5.4}
$$

Dal punto di vista funzionale, i ricercatori di CORELS hanno effettuato dei test preliminari (non presenti nel documento) che attestano la superiorità della *curiosity* rispetto al *lower bound* su alcuni problemi specifici. Non sembra dunque che questa policy offra un miglioramento generale rispetto a quelle implementate precedentemente, d'altro canto, implementare altre curious functions potrebbe essere, a detta dei ricercatori di CORELS, un problema di forte interesse.

Chapter 6

Esperimenti

6.1 Scopo dell'esperimento

Utilizzando i dati forniti da [5], si vuole sfruttare CORELS come esploratore delle relazioni tra le alterazioni genetiche del paziente e la tipologia tumorale a cui sono affetti. Il vantaggio di un modello ML in questo contesto è evidente se messo nella prospettiva della mole di dati su cui occorre lavorare e della difficoltà, pratica ed economica, nel reperire queste informazioni. Data la limitazione che CORELS ci impone, attraverso l'uso di label binarie, in questo documento verrà implementato in modo tale da distinguere due tipologie tumorali. Si vedrà che molte delle regole trovate da CORELS evidenziano quelli che vengono conosciuti comunemente come **marker tumorali**, di cui si parlerà successivamente.

6.2 Ambiente d'esecuzione

Tutti gli esperimenti sono stati effettuati su un laptop con un Intel® Core™ i7-8750H (9M cache, 2.20GHz) e 16 GB RAM.

6.3 Approccio ai dati

Per effettuare gli esperimenti sono stati utilizzati i dati di due file presenti in [5]: *hotnet2-samples.txt* e *snvs_strictly_filtered.tsv*. Durante la prima fase, l'obiettivo è stato la conversione dei dati forniti in formato binario.

Nel file *hotnet2-samples.txt* viene fornita la **label** del nostro dataset, ovvero la tipologia di tumore presente nei pazienti. È stato necessario però, filtrare i dati, in modo da avere solo i pazienti con le due tipologie tumorali prese in considerazione. La prima tipologia assumerà valore 1 e la seconda valore 0, in questo modo la label è nel formato binario.

Per quanto riguarda le mutazioni subite da ogni paziente, presenti nell'altro file, la conversione è leggermente diversa. L'idea alla base è quella di utilizzare come *features* tutti i geni mutati, indicando con 1 la loro presenza nel paziente e con 0 la loro assenza. Per questa procedura, si è sfruttato un dizionario contenente i geni mutati come chiave e la lista dei pazienti possessori (la riga del file) come valore. Sfruttando questa struttura dati si è riusciti a creare un vero e proprio dataset binario dei geni mutati.

Dopo le procedure descritte sopra, e un'operazione di *join* tra le tabelle ricavate dai due file, il nostro dataset finale è binario e utilizzabile da CORELS.

Il dataset binario è abbastanza preciso e ragionevole per il tipo di problema che si vuole risolvere. Occorre notare però, che l'ordine di grandezza delle feature nel dataset preso in analisi è notevolmente più amplio dei dataset utilizzati negli esperimenti di CORELS. Per questa ragione è stato necessario implementare un sistema di **pruning** delle mutazioni, in modo tale da poter diminuire il volume dei dati, cercando di eliminare le mutazioni meno frequenti di una certa percentuale all'interno del dataset, assumendo che non saranno presenti in nessuna regola ottima.

nome esperimento	pazienti	geni mutati
LUAD o LUSC	348	14.463
OV o COADREAD	509	11.114

Table 6.1: Quantità di pazienti (righe) e mutazioni (colonne) dei dataset binari ricavati attraverso i dati presenti in [5]

6.4 Implementazione

Tutto il codice prodotto[6] per i test è stato sviluppato su *Python 3.10*. I test che verranno mostrati sono stati realizzati attraverso la libreria Python di CORELS (**pycorels**)[2]. Per *Python 3.10*, allo stato attuale, per problemi di compatibilità con la libreria *numpy*, pycorels non sarà immediatamente utilizzabile seguendo la guida all'installazione. La causa di questo problema è la deprecazione da parte di *numpy* dell'attributo **numpy.bool**, sostituito da **numpy.bool_**. Per risolvere occorre aggiornare il tipo di variabile manualmente nei seguenti file:

- file: *corels.py*; riga 276
- file: *utils.py*; riga 17

6.5 Parametri modificabili degli esperimenti

Durante gli esperimenti sono stati modificati vari parametri per osservare l'effetto sulla qualità e la funzionalità dell'algoritmo

Attributo	**Descrizione**
percentage	Questo attributo permette l'attività di **pruning** del dataset. Se per esempio questo attributo assumesse valore 0.01 vorrà dire che vengono contate solo le mutazioni presenti come minimo nell'1% dei pazienti
node	indica il numero massimo di nodi espandibili dal Trie, questo fattore è importante per il numero di regole che CORELS riuscirà a valutare
policy	Con questo attributo viene selezionato il tipo di policy di ricerca che verrà applicata. Le policy implementate sono descritte nel capitolo precedente
min_support	Questo parametro vincola le regole all'interno della rule list a comprendere un certo numero di pazienti. Se per esempio l'attributo assumesse valore 0.3 vorrà dire che ogni regola prodotta da CORELS deve includere almeno il 30% dei pazienti

Table 6.2: Lista degli attributi modificabili sul codice di testing

6.6 Effetti del Pruning sulle regole

Quanto cambia lo score di CORELS se vengono eliminate alcune mutazioni dal dataset in relazione alla loro frequenza? In questa prima parte di esperimento si vuole provare quanto eliminare mutazioni a una certa frequenza, alteri lo score finale. L'ipotesi è la seguente: *una mutazione frequente su una percentuale estremamente ridotta del dataset, non verrà mai inclusa nelle regole ottime, in quanto non abbastanza rilevante da formarne una.* A riprova di ciò sono stati fatti diversi test a parità di nodi (100000), policy di ricerca (*Objective*) e min_support (0.01):

percentuale	score	mutazioni
0.01	0.9236	1115
0.05	0.9401	5
0.10	0.9374	4
0.20	0.9431	3
0.30	0.9187	2
0.40	0.9205	2

(a) test OV-COADREAD

percentuale	score	mutazioni
0.01	0.6884	7593
0.05	0.6800	208
0.10	0.6685	8
0.20	0.6558	2
0.30	0.6529	2
0.40	0.6829	1

(b) test LUAD-LUSC

I dati confermano l'idea: nonostante il numero di mutazioni sia estremamente minore di tutte quelle disponibili, lo score del modello non viene fortemente intaccato. Per gli esperimenti successivi fissiamo il valore a **0.01** per il dataset OV o COADREAD e **0.05** in quello LUAD e LUSC

6.7 LUAD o LUSC: risultati esperimento

Il primo esperimento ha preso in considerazione come prima tipologia tumorale **LUAD** (adenocarcinoma polmonare) e come seconda **LUSC** (carcinoma polmonare squamoso). Il carcinoma polmonare squamoso è una variante del adenocarcinoma polmonare, perciò condivideranno alcuni geni mutati.

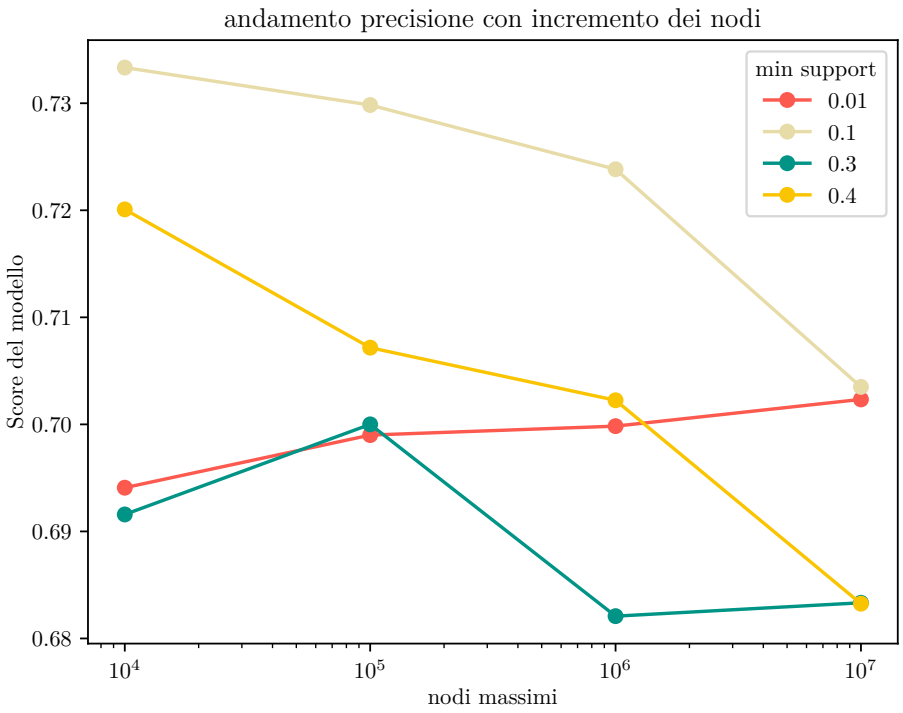


Figure 6.1: Grafico che analizza la precisione in base al numero di nodi

In media il modello, indipendentemente da tutti i parametri, ha uno score del **70**%. Viene premesso che ogni esecuzione viene interrotta a causa del superamento dei nodi. La ragione dietro la decrescita vista nel grafico potrebbe essere risultato di diversi fattori: in primo luogo potrebbe dipendere dalle policy di ricerca che espandono l'albero in maniera differente, successivamente il modello potrebbe andare in **overfitting** nonostante un numero ridotto di geni presi in considerazione. Occorre ricordare che queste mutazioni sono di tipo **SNV**, dunque le mutazioni sono estremamente rare. Si evidenzi anche che il training set e il test set cambiano di test in test, per cui le regole possono alterare. Ora prendiamo una delle regole trovate da CORELS, è interessante vedere la presenza di *KRAS* che è un gene proto oncogeno, e del *TP53* un gene oncosoppressore che interviene in molti meccanismi antitumorali.

if $KRAS$ **and NOT** $ARID1A$ **then predict** $LUAD$
else if $TP53$ **and NOT** $EGFR$ **then predict** $LUSC$
else predict $LUAD$

Figure 6.2: rule list ottimale identificata da CORELS, con parametri: nodi: 100000; score: 0.843; min_support:0.1

6.8 OV o COADREAD: risultati esperimento

Il secondo esperimento invece prende due tumori in zone distinte: **COADREAD** (carcinoma del colon - retto) e **OV** (carcinoma ovarico). Come si vedrà dai dati, lo score è molto più alto del test precedente, questo perché i due tumori hanno maggiori geni differenti. Valgono anche per questo test i presupposti del precedente.

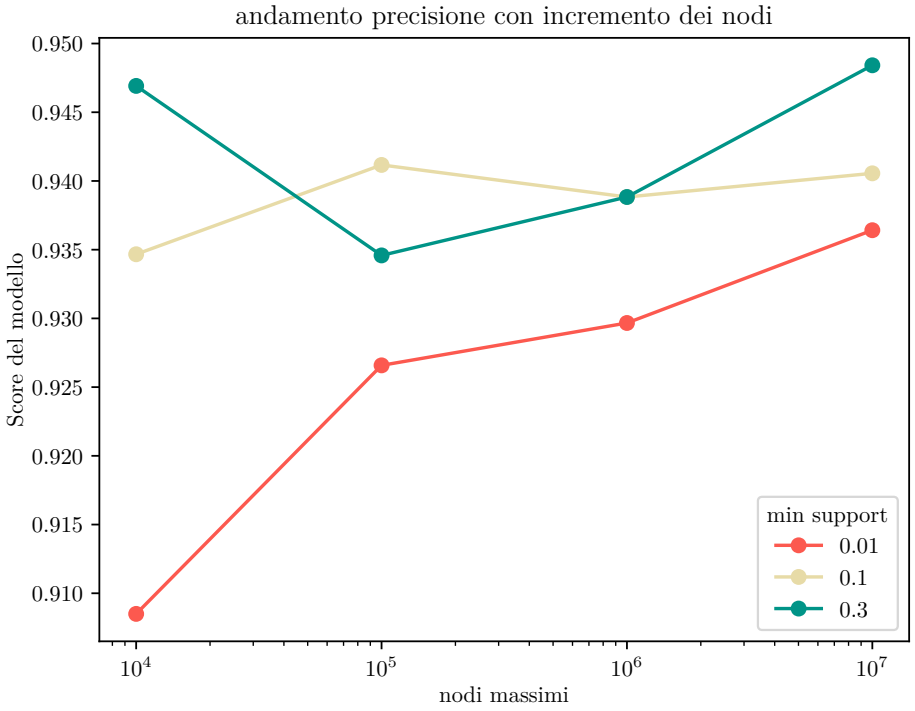


Figure 6.3: Grafico che analizza la precisione in base al numero di nodi

Si noti che in questo caso il *min_support* a 0.4 non è presente nel grafico, questo perché lo score è estremamente più basso e impedisce una lettura pulita dei dati. L'andamento sembra essere opposto a quello precedente, essendo molto diversi, l'espansione dei nodi potrebbe trovare altri geni differenti, in modo da essere più precisi. C'è da ricordare che lo score ha un fattore aleatorio in base alla divisione del dataset. In media il mod-

ello, indipendentemente dai parametri, ha uno score dell'80%. Lo score viene fortemente abbassato dallo score dei test con *min_support* a 0.4 che vediamo nel grafico sotto

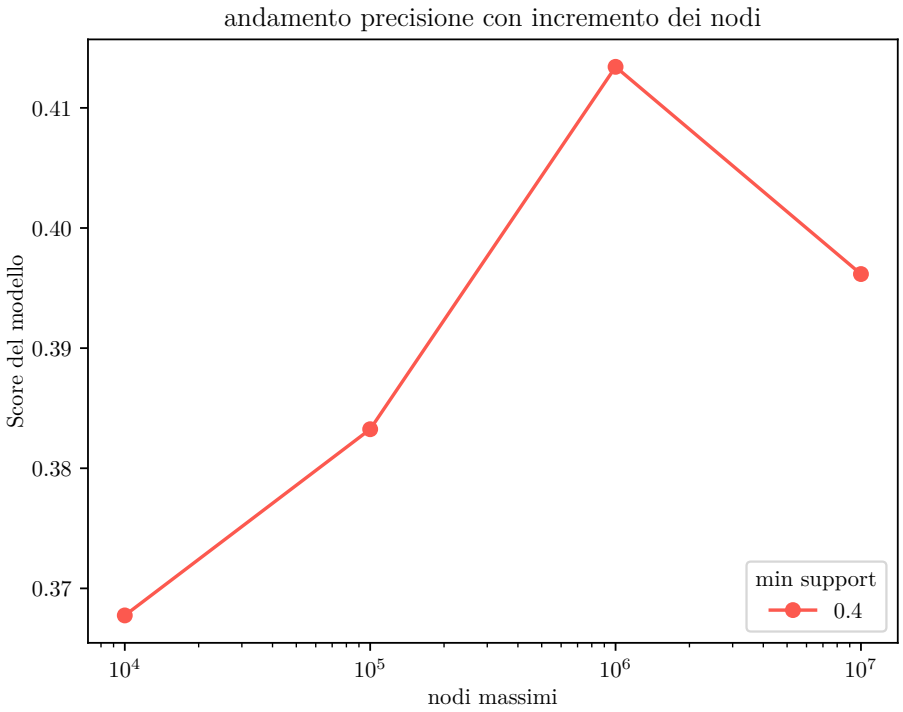


Figure 6.4: Grafico che analizza la precisione in base al numero di nodi per un min_support a 0.4

Anche in questo contesto è interessante visualizzare una delle regole prodotte da CORELS in questo esperimento per vedere se sono presenti geni noti.

if NOT *APC* **and NOT** *KRAS* **then predict** *OV*
else predict *LUSC*

Figure 6.5: rule list ottimale identificata da CORELS, con parametri: nodi: 100000; score: 0.99; min_support:0.1

La prima cosa che si nota è che la regola è più semplice di quella precedente. *APC* e *KRAS* sono già noti come geni che se alterati portano allo sviluppo di un carcinoma del colon-retto[11].

6.9 Comparazione rule lists e decision trees

Per ogni test effettuato con CORELS, si è voluto effettuare un ulteriore test con un altro modello IML, ovvero i *decision trees*. È stata sfruttata la libreria *scikit-learn*[7] in cui è stato implementato il modello decision tree (DT).

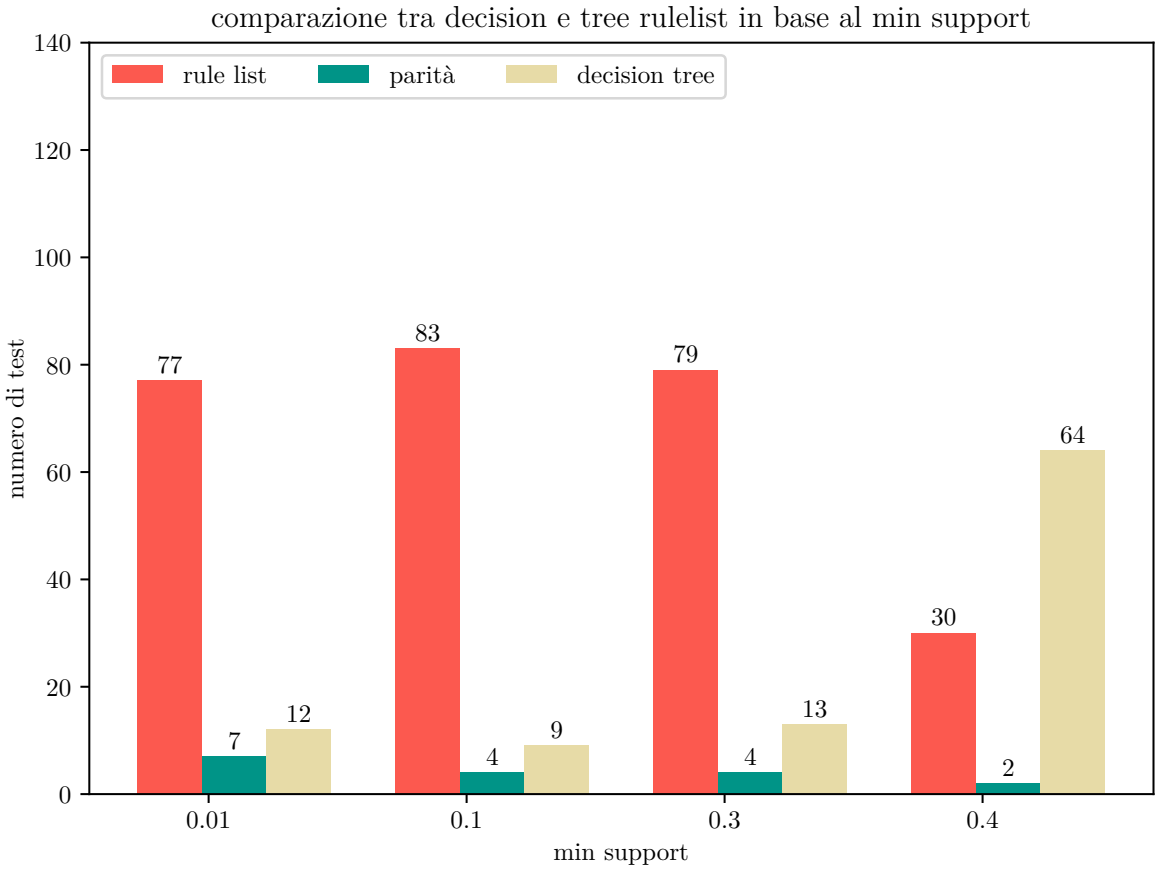
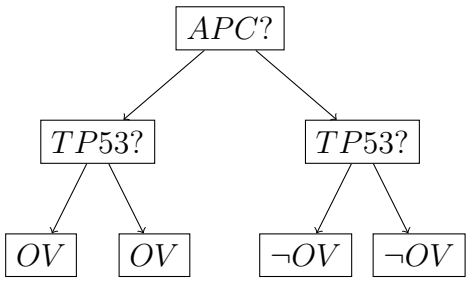


Figure 6.6: Grafico che compara il numero di volte in cui un modello supera l'altro in precisione. Fatto su 96 test per min_support

Nella fase sperimentale si è notato che il parametro min_support, se troppo elevato, rende le rule lists fortemente meno efficaci dei DT. La ragione dietro questo comportamento è riconducibile al fatto che, sviluppare una regola che includa il 40% del dataset e che sia molto precisa, è estremamente difficile. Imporre il min_support dunque, impedisce lo sviluppo normale di ricerca delle regole ottime di CORELS. Compariamo visivamente l'output di un test con rule list e DT (OV o COADREAD con percentuale di pruning del 5% e min_support del 10%), si ricordi che la lettura dell'albero va fatta in modo tale che, **true** si trovi a destra e **false** a sinistra.



if APC **then predict** $COADREAD$
else predict OV

(a) Decision Tree. Score: 0.912 (91%). **(b)** Rule List. Score: 0.922 (92%)

6.10 Conclusioni

CORELS è un algoritmo versatile e funzionale per produrre ottime rule lists. Nel complesso una rule list è di più facile lettura di un decision tree, il che lo rende un perfetto strumento diagnostico. Nel nostro contesto è stato efficace a evidenziare alcune mutazioni genetiche risultano correlate con le tipologie tumorali considerate, con risultati e score soddisfacenti. Qualora la ricerca porterà a una versione di CORELS utilizzabile su dataset non binari, un interessante sviluppo futuro sarà considerare label categoriche e continue, ad esempio per predire diverse tipologie tumorali o misurazioni cliniche di pazienti.

Linkografia

[1] CORELS. *CORELS library in c++*. URL: https://github.com/corels/corels/tree/master.

[2] CORELS. *CORELS library in python (pycorels)*. URL: https://github.com/corels/pycorels.

[3] *JUDEA PEARL ACM Turing Award page*. URL: https://amturing.acm.org/award_winners/pearl_2658896.cfm.

[4] UCI Machine Learning. *Student Alcohol Consumption: Social, gender and study data from secondary school students*. URL: https://www.kaggle.com/datasets/uciml/student-alcohol-consumption.

[5] Mark D. M. Leiserson et al. *3110 samples analyzed by HotNet2 with Pan-Cancer mutation data from [9]. Site of Paper [12]*. URL: http://compbio-research.cs.brown.edu/pancancer/hotnet2/#!/data.

[6] Riccardo Modolo. *Codice Tesi*. URL: https://github.com/RickSrick/rulelist-cancer-CORELS.

[14] Christoph Molnar. *Interpretable Machine Learning, A Guide for Making Black Box Models Explainable*. 2021. URL: https://christophm.github.io/interpretable-ml-book/.

[7] *scikit-learn Machine Learning in Python*. URL: https://scikit-learn.org/stable/index.html.

Bibliografia

[8] Elaine Angelino et al. “Learning Certifiably Optimal Rule Lists for Categorical Data”. In: *J. Mach. Learn. Res.* 18.1 (Jan. 2017), pp. 8753–8830. ISSN: 1532-4435. URL: https://www.jmlr.org/papers/volume18/17-716/17-716.pdf.

[9] Kyle Chang et al. “The Cancer Genome Atlas Pan-Cancer analysis project”. In: *Nature Genetics* 45.10 (Oct. 2013), pp. 1113–1120. ISSN: 1546-1718. DOI: 10.1038/ng.2764. URL: https://doi.org/10.1038/ng.2764.

[10] Jens Clausen. “Branch and Bound Algorithms - Principles and Examples.” In: (1999). URL: https://web.archive.org/web/20150923214803/http://www.diku.dk/OLD/undervisning/2003e/datV-optimer/JensClausenNoter.pdf.

[11] Joshua H. Cook et al. “The origins and genetic interactions of KRAS mutations are allele- and tissue-specific”. In: *Nature Communications* 12.1 (Mar. 2021), p. 1808. ISSN: 2041-1723. DOI: 10.1038/s41467-021-22125-z. URL: https://doi.org/10.1038/s41467-021-22125-z.

[12] Mark D. M. Leiserson et al. “Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes”. In: *Nature Genetics* 47.2 (Feb. 2015), pp. 106–114. ISSN: 1546-1718. DOI: 10.1038/ng.3168. URL: https://doi.org/10.1038/ng.3168.

[13] Tim Miller. “Explanation in artificial intelligence: Insights from the social sciences”. In: *Artificial Intelligence* 267 (2019), pp. 1–38. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2018.07.007.

Ringraziamenti

In primo luogo, mi preme ringraziare il professor Pellegrina e il professor Vandin per il loro essenziale contributo alla redazione di questo documento, attraverso le loro acute direttive e il loro supporto.

In secondo luogo, ci tengo a ringraziare la mia famiglia, i quali hanno creduto in me molto di più di quanto abbia fatto io nella mia vita.

Ultimi ma non meno importanti, gli incredibili legami creati in questi anni: non credo sarò mai fiero dei miei traguardi quanto della mia capacità di scegliere delle persone meravigliose:

Matteo	Francesco	Christian
Kabir	Luca	Ippolito
Marius	Gallo	Francesco
Alessandro	Diletta	Angela
Sara	Anna	Giulia
Giada	Marika	Matteo
Leonardo	Gianmarco	Alessandro

per citarne alcuni. Auguro a tutti voi la felicità più totale e a me di poterla condividere con voi.