UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA E
DELL'AUTOMAZIONE

Classificazione Pulsar

Autori:
Sonaglioni Matteo
Beccerica Sara
Hoxha Emanuel

Anno Accademico 2024/2025

Indice

Sommario

L'obiettivo del progetto è sviluppare un programma integrato che utilizzi Python e Prolog per la classificazione di pulsar all'interno di un dataset di stelle di neutroni. In particolare, verranno implementate tecniche di apprendimento supervisionato per creare un albero di decisione basato su un set di dati di addestramento, per poi utilizzarlo per la classificazione di nuovi oggetti.

1 Dataset

1.1 Informazioni principali

Il dataset utilizzato per il progetto è stato selezionato da Kaggle e si intitola ”Predicting Pulsar Star”. Questo dataset contiene misurazioni provenienti da osservazioni astronomiche, con l'obiettivo di distinguere i pulsar veri dal rumore di fondo. Infatti, le pulsar sono delle rare stelle di neutroni che producono emissioni radio particolari. Ogni candidato è descritto da 8 variabili continue e una singola variabile che rappresenta la classe (quindi se è una pulsar o meno). Le altre variabili sono:

- ⋆ Mean of the integrated profile
- ⋆ Standard deviation of the integrated profile
- ⋆ Excess kurtosis of the integrated profile
- ⋆ Skewness of the integrated profile
- ⋆ Mean of the DM-SNR curve
- ⋆ Standard deviation of the DM-SNR curve
- ⋆ Excess kurtosis of the DM-SNR curve
- ⋆ Skewness of the DM-SNR curve

Come si può notare, le variabili sono suddivise in due macro-categorie:

- *Profilo integrato dell'impulso:* ogni pulsar produce un modello unico di emissione dell'impulso. È come un'impronta digitale del pulsar, quindi è possibile identificarli tramite questo attributo. Tuttavia, questo profilo varia leggermente in ogni periodo, rendendoli più difficili da rilevare. Ciò accade perché i loro segnali non sono uniformi e non sono completamente stabili nel tempo. Ad ogni modo, questi profili diventano stabili quando vengono mediati su molte migliaia di rotazioni.
- *Curva DM-SNR:* le onde radio emesse dai pulsar raggiungono la Terra dopo aver viaggiato a lungo nello spazio, che è pieno di elettroni liberi. Poiché le onde radio sono di natura elettromagnetica, interagiscono con questi elettroni, e questa interazione rallenta le onde. È importante notare che i pulsar emettono un'ampia gamma di frequenze, e la quantità

con cui gli elettroni rallentano l'onda dipende dalla frequenza. Le onde con frequenza più alta vengono rallentate meno rispetto a quelle con frequenza più bassa. In altre parole, le frequenze più basse raggiungono il telescopio più tardi rispetto a quelle più alte. Questo fenomeno è chiamato dispersione.

Per quanto riguarda il significato dei parametri misurati:

1. *Mean:* è la media, ovvero la somma di tutti i valori in un insieme di dati divisa per il numero totale dei valori.

$$Mean = \frac{\sum_{i=1}^{n} x_i}{n}$$

2. *Standard Deviation:* la deviazione standard misura quanto i valori di un dataset sono sparsi rispetto alla media. Una deviazione più alta indica che i dati sono più dispersi.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}}$$

3. *Excess Kurtosis:* la curtosi descrive la "forma" delle code di una distribuzione rispetto ad una distribuzione normale. Una curtosi alta (maggiore di 0) indica code più spesse e più pesanti, ossia una maggiore probabilità di valori estremi (più "acuminata"). Una curtosi bassa (minore di 0) indica code più sottili e meno pesanti (più "piatta"). Il -3 è sottratto per centrare la curtosi intorno a 0 in distribuzioni normali.

$$Kurtosis = \frac{n \sum (x_i - \mu)^4}{(\sum (x_i - \mu)^2)^2} - 3$$

4. *Skewness:* l'asimmetria misura quanto una distribuzione è simmetrica rispetto alla media. Una skewness positiva indica che la coda destra è più lunga o i valori sono più comuni. Una skewness negativa indica che la coda sinistra è più lunga o i valori più piccoli sono più comuni.

$$Skewness = \frac{\frac{1}{n} \sum (x_i - \mu)^3}{\sigma^3}$$

Riepilogo del dataset:

- 17.898 esempi totali;
- 1.639 esempi positivi;
- 16.259 esempi negativi.

1.2 Pulizia del dataset

Il dataset analizzato presenta una significativa asimmetria nella distribuzione delle classi, il che richiede una gestione accurata per garantire che i risultati dell'apprendimento automatico siano affidabili e rappresentativi. Per affrontare questa problematica, è indispensabile innanzitutto condurre un'analisi preliminare approfondita che garantisca la qualità e l'affidabilità del dataset. In particolare, occorre:

- *Gestione dei dati nulli:* verificare che il dataset non contenga valori mancanti o nulli. Qualora presenti, tali valori devono essere eliminati o trattati attraverso tecniche appropriate, come l'imputazione basata su valori medi, mediani o modelli predittivi.

- *Eliminazione di esempi duplicati:* identificare ed eliminare eventuali record duplicati, che potrebbero introdurre ridondanza e distorcere le analisi o il processo di apprendimento.

- *Identificazione e trattamento degli outlier:* individuare i valori anomali o estremi attraverso strumenti statistici o tecniche grafiche (ad esempio utilizzando i boxplot). Gli outlier possono essere gestiti eliminandoli o trasformandoli, a seconda del loro impatto sul contesto del problema.

Questi passaggi sono essenziali per assicurare che il dataset utilizzato sia di alta qualità e possa supportare un'analisi accurata e l'addestramento efficace dei modelli di apprendimento automatico. Un dataset ben preparato non solo migliora le prestazioni del modello, ma contribuisce anche a ridurre il rischio di interpretazioni errate o di decisioni basate su dati non rappresentativi.

2 Apprendimento

2.1 Suddivisione del dataset

Nel codice, la suddivisione viene effettuata utilizzando il metodo di *validation*, sia in Python che in Prolog: il dataset viene suddiviso in *chunks* tramite la libreria ”StratifiedKFold”, che bilancia i chunks a seconda dei valori definiti nella colonna ”target_class”. Inoltre, la funzione ”train_test_split” suddivide i chunks in due insiemi:

- *Set di training (80%):* utilizzato per addestrare il modello;
- *Set di test (20%):* utilizzato per valutare le prestazioni del modello.

La funzione ”export_to_prolog” è responsabile della creazione dei file Prolog necessari per addestrare e testare il modello di apprendimento automatico direttamente in Prolog. La funzione itera su ogni esempio del dataset, dove le feature e la classe target vengono formattate in una clausola Prolog:

- Per il set di training: *dtrain([feature1, feature2, ...], label).*
- Per il set di test: *dtest([feature1, feature2, ...], label).*

Quindi, il programma itera per ogni chunks la fase di apprendimento e test, recupera le statistiche ottenute (Matrice di confusione, Accuratezza, Precisione, Recall, Errore e F1-Score), per poi calcolarne una media e restituire i risultati.

2.2 Induzione dell’albero

Per l’induzione dell’albero sono stati utilizzati due criteri differenti:

- **Gini**: è una misura della probabilità che un elemento scelto dal dataset sia classificato erroneamente se fosse etichettato in base alla distribuzione delle classi nel nodo. Quindi, valori vicini allo 0 indicano che tutti gli esempi appartengono alla stessa classe (purezza massima), mentre valori vicini a 0.5 indicano una distribuzione uniforme tra le classi (impurezza massima).

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2$$

- **Entropia**: misura l'incertezza o impurità in un sistema. Rappresenta la quantità di imprevedibilità nella distribuzione delle classi all'interno di un nodo. Quindi, valori vicino allo 0 indicano bassa incertezza (alta purezza), mentre valori più alti indicano maggiore incertezza (impurezza).

$$Entropia(S) = -\sum_{i=1}^{n} p_i log_2 p_i$$

Le differenze tra Gini ed Entropia sono:

- *Intervallo di valori:* il criterio di Gini varia tra 0 e 0.5 per due classi, mentre l'entropia varia tra 0 ed 1.
- *Sensibilità:* l'impurezza di Gini è meno severa rispetto all'entropia. Questo significa che l'entropia tende a preferire nodi più puri.
- *Calcolo:* il criterio di Gini è più semplice da calcolare rispetto all'entropia poiché non coinvolge logaritmi.

2.3 Analisi del codice

Il codice in Prolog implementa un algoritmo di induzione di alberi di decisione. Di seguito, si analizza il codice passo per passo.

Vengono caricati i file "train_data.pl" e "test_data.pl", che contengono i dati di addestramento e di test.
La funzione principale è "run_tree", che permette di leggere i dati di training e test, costruisce un albero decisionale e valuta l'albero sui dati di test, calcolando l'accuratezza e la matrice di confusione (Veri positivi, Falsi positivi, Veri negativi, Falsi negativi).
L'albero decisionale è costruito in modo ricorsivo:

1. *Foglie:* quando tutti i dati del nodo appartengono alla stessa classe.
2. *Nodi interni:* quando si effettua una suddivisione dei dati basata su un attributo ed una soglia.

Il calcolo del guadagno informativo viene utilizzato per determinare la qualità di una suddivisione, ed è appunto basato sull'Entropia o sul criterio di Gini. Il predicato "best_split" esplora tutte le possibili combinazioni di *Indice dell'attributo* e *Soglie*, calcolando il guadagno informativo per ogni suddivisione e scegliendo quella con il guadagno massimo.

Il predicato ”predict” naviga l'albero decisionale dai nodi alla foglia, basandosi sul valore dell'attributo e sulla soglia, per produrre una predizione.
Per la valutazione del modello si comparano le predizioni dell'albero con le classi reali dei dati di test e ne calcola l'accuratezza:

$$Accuracy = \frac{Previsioni\_corrette}{Totale\_previsioni}$$

La matrice di confusione quantifica la performance del modello:

1. True positive (TP): previste positive e realmente positive;
2. True negatives (TN): previste negative e realmente negative;
3. False positive (FP): previste positive ma realmente negative;
4. False negatives (FN): previste negative ma realmente positive.

Inoltre, sono presenti alcune funzioni di supporto, come:

1. split: per suddividere i dati in due insiemi (sinistra e destra) in base ad un attributo ed una soglia;
2. count: conta la frequenza di un valore in una lista.

Il codice sfrutta ampiamente la ricorsione per costruire l'albero, navigarlo e valutare i dati. Inoltre, la suddivisione dei nodi utilizza solo soglie che migliorano l'omogeneità delle classi nei sottoinsiemi. Infine, si evita la divisione con insiemi vuoti durante la costruzione dell'albero.
I limiti di questo algoritmo sono l'efficienza, poiché la ricerca della suddivisione può essere costosa per dataset con molti attributi o soglie, e l'overfitting, poiché l'albero potrebbe adattarsi troppo ai dati di training. Sarà necessario applicare il meccanismo di potatura dell'albero per il codice in Prolog.
Lo stesso algoritmo è stato implementato in Python, con delle leggere differenze in riferimento alla sintassi del linguaggio:

1. **Inizializzazione e parametri:** il costruttore della classe *(__init__)* consente di configurare diversi iperparametri:
   (a) *max_depth:* profondità massima dell'albero;
   (b) *min_samples_leaf:* numero minimo di campioni richiesti per un nodo foglia;

(c) *min_information_gain:* guadagno informativo minimo per permettere uno split;

(d) *numb_of_features_splitting:* numero di feature da considerare in fase di splitting (sqrt, log, o tutte);

(e) *amount_of_say:* peso del classificatore per integrazione con algoritmi ensemble come AdaBoost.

2. **Splitting:** divide i dati in due gruppi in base ad una soglia per una specifica feature, sceglie casualmente un sottoinsieme di feature da considerare per lo splitting (basato sull'impostazione di *numb_of_features_splitting*), infine identifica il miglior split possibile per un dataset minimizzando l'entropia/massimizzando Gini di partizione.

3. **Creazione dell'albero:** la funzione *_create_tree* costruisce l'albero decisionale in modo ricorsivo: si interrompe se viene raggiunta la profondità massima o se non si soddisfano i criteri di split. Ogni nodo memorizza informazioni sui dati, la feature utilizzata per lo splitting e le probabilità di classe.

4. **Predizione:** la funzione *_predict_one_sample* predice la probabilità delle classi per un singolo campione, navigando attraverso i nodi dell'albero fino ad una foglia. Si calcola poi le probabilità delle classi per un insieme di dati e si restituisce un'etichetta della classe con la probabilità più alta per ogni campione. L'importanza di ciascuna feature viene valutata visitando ricorsivamente i nodi dell'albero.

La classe *TreeNode*, invece, implementa la struttura di un nodo utilizzato nell'albero decisionale. Ogni nodo può rappresentare un nodo interno (che effettua uno split sui dati) o un nodo foglia (che fornisce la predizione per una classe).
I vantaggi di questo algoritmo sono la modularità, poiché ogni componente è implementato in modo chiaro ed indipendente, e la flessibilità, supportando personalizzazioni attraverso iperparametri.
I limiti invece sono la complessità, che può aumentare per dataset con molte feature o elevata profondità, e non implementa tecniche di pruning per evitare l'overfitting.

2.4 Potatura

La potatura è un processo che mira a ridurre la complessità di un albero decisionale eliminando nodi o sottoalberi che non migliorano significativamente le

prestazioni, rendendolo più generale e meno incline all'overfitting.
La potatura è stata implementata solo per gli algoritmi in Prolog, che sono molto meno efficienti rispetto a quelli in Python. Il processo della potatura è il seguente:

1. Ogni nodo dell'albero viene valutato:
   (a) Viene calcolata l'accuratezza dell'albero completo;
   (b) Viene calcolata l'accuratezza che si otterrebbe trasformando quel nodo in una foglia.
2. Se la potatura non riduce l'accuratezza, il nodo viene trasformato in una foglia.
3. Questo processo continua ricorsivamente fino a valutare tutti i nodi dell'albero.

Il caso base è quello di un nodo foglia che rimane invariato durante la potatura. Per un nodo:

- I suoi sottoalberi (sinistro e destro) vengono potati ricorsivamente.
- Dopo la potatura, il sistema valuta se il nodo può essere convertito in una foglia usando il predicato *should_prune*.
- Se il nodo è da potare, viene trasformato in una foglia tramite il predicato *convert_to_leaf*.

Un sottoalbero viene potato solo se l'accuratezza del modello dopo la potatura (calcolata trasformando il nodo in una foglia) è uguale o superiore a quella dell'albero completo. Inoltre, un nodo viene convertito in una foglia utilizzando la classe di maggioranza delle etichette presenti nei suoi sottoalberi, che vengono raccolte ricorsivamente. Per calcolare la classe di maggioranza, il predicato *majority_class*:

- Ordina le etichette;
- Raggruppa etichette uguali;
- Trova la classe con il massimo numero di occorrenze.

L'accuratezza dell'albero è calcolata testando sui dati di test. I vantaggi della potatura sono sicuramente la riduzione dell'overfitting, poiché l'albero diventa meno specifico ai dati di training, l'efficienza, perché un albero più piccolo è più rapido da attraversare, e l'interpretabilità, ovvero alberi meno profondi sono più facili da comprendere.

3 Classificazione

La classificazione di un nuovo elemento nell'apprendimento automatico si riferisce al processo di assegnare un'etichetta (o classe) ad un insieme di caratteristiche (features) in base ad un modello già addestrato, che utilizza la sua struttura per associare l'esempio ad una delle classi. Ad esempio, su un albero decisionale, partendo dalla radice, si percorrono i nodi in base ai valori delle caratteristiche dell'esempio, fino a raggiungere una foglia, che contiene la classe predetta. L'importanza della classificazione fa riferimento ad applicazioni pratiche, come la predizione di malattie su pazienti, rilevamento di frodi, filtraggio di spam, riconoscimento di oggetti o volti in un'immagine.

3.1 Classificazione di un nuovo esempio

La classificazione viene effettuata in Python utilizzando il modello Entropia passando i dati in input e analizzando l'albero di decisione.

3.2 Interfaccia grafica

Per semplificare la visualizzazione delle statistiche e la classificazione di un nuovo elemento, è stata implementata un'interfaccia grafica che restituisce i risultati ottenuti dopo l'apprendimento. L'interfaccia è suddivisa in due parti:

- **Statistiche:** Restituisce le statistiche ottenute dall'apprendimento in Python e Prolog, differenziandole per ogni criterio utilizzato (Gini ed Entropia).
- **Classificazione:** Permette di inserire in input dei dati (veritieri o meno) e si classifica il nuovo oggetto in Pulsar o Stella di neutroni.

4 Risultati

Avendo un dataset molto grande e gli algoritmi utilizzati possono analizzare solo un numero limitato di elementi alla volta, bisogna fare alcune considerazioni sulle regole create dagli alberi di decisione.
Suddividere in batch potrebbe portare a perdere alcune regole globali, specialmente se le relazioni tra i dati emergono solo a livello di dataset completo. Una soluzione richiederebbe un approccio basato sull'apprendimento parziale su ogni chunk, l'unione delle regole apprese senza duplicati e senza conflitti, basandosi su criteri come la fiducia più alta *(Confidence-based Selection)* o la preferenza di specificità delle regole.
Inoltre, effettuare la media dei risultati di ciascun blocco potrebbe non essere sufficiente per rappresentare correttamente l'efficacia del modello, soprattutto quando si ha a che fare con dataset sbilanciati o con un set di regole complesso. Una soluzione potrebbe essere quella di calcolare la matrice di confusione complessiva per tutti i blocchi aggregati e successivamente calcolarne le statistiche globali: in questo modo non si perde nessuna informazione, poiché combinando le matrici di confusione dei vari blocchi non si perde nessuna regola.
Per dataset molto grandi e sbilanciati, oltre alla strategia dell'elaborazione a blocchi, si possono adottare diverse strategie:

- **Campionamento (Sampling):** si seleziona casualmente un sottoinsieme di dati rappresentativo, mantenendo la proporzione tra le classi;
- **Mini-batch Stochastic Gradient Descent (SDG):** utilizza algoritmi che supportano l'elaborazione in mini-batch, comuni in modelli di machine learning come le reti neurali o la regressione logistica;
- **Clusterizzazione:** utilizza tecniche per ridurre la dimensione del dataset selezionando rappresentanti di cluster;
- **Bilanciamento incrementale:** applica SMOTE in modo incrementale su batch di dati per generare solo i campioni necessari in ogni blocco;

Considerazioni:
Effettuando dei test è stata implementata una tecnica come SMOTEEN, che genera nuovi campioni sintetici per la classe minoritaria, rendendo più rappresentativa la distribuzione, e rimuove i campioni rumorosi o mal classificati, migliorando la qualità del set. In questo caso è stato prima diviso il dataset originale in training e test, poi viene applicato SMOTEENN solo al training set per evitare di influenzare negativamente la valutazione del modello. I risultati ottenuti non sono stati soddisfacenti: negli alberi decisionali implementati in Python l'accuratezza si è mantenuta su livelli simili rispetto a quelli senza bilanciamento,

mentre negli alberi basati sull'entropia in Prolog l'accuratezza è scesa al di sotto dell'80%.

4.1 Statistiche

Di seguito sono riportati i risultati della classificazione, includendo l'accuratezza, la precisione, il ricavo, l'F1-score, l'errore e la matrice di confusione degli alberi decisionali implementati in Python e Prolog. Vengono evidenziate le differenze tra l'utilizzo del criterio di Gini e quello dell'Entropia.

Modello: Gini (Python)	**Modello: Gini (Prolog)**
Accuratezza: 0.9174 **Precisione:** 0.0926 **Ricavo:** 0.0975 **F1-Score:** 0.0949 **Errore:** 0.0826 **Matrice di Confusione:** TN: 3265 FP: 4 FN: 292 TP: 21	**Accuratezza:** 0.9126 **Precisione:** 0.0926 **Ricavo:** 0.0975 **F1-Score:** 0.0949 **Errore:** 0.0874 **Matrice di Confusione:** TN: 3269 FP: 0 FN: 313 TP: 0
Modello: Entropia (Python)	**Modello: Entropia (Prolog)**
Accuratezza: 0.9732 **Precisione:** 0.8789 **Ricavo:** 0.8117 **F1-Score:** 0.8383 **Errore:** 0.0268 **Matrice di Confusione:** TN: 3232 FP: 37 FN: 59 TP: 254	**Accuratezza:** 0.9098 **Precisione:** 0.8789 **Ricavo:** 0.8117 **F1-Score:** 0.8383 **Errore:** 0.0902 **Matrice di Confusione:** TN: 3119 FP: 150 FN: 173 TP: 140

4.2 Conclusioni

I risultati appresi devono tenere conto del fatto che per gli alberi decisionali in Prolog è stata implementata la potatura dell'albero, che dovrebbe aumentarne

l'efficienza e l'efficacia, mentre per Python non sono stati implementati. Inoltre, l'algoritmo in Python non avrebbe avuto le stesse limitazioni dimensionali di Prolog, che necessita un calcolo di circa 1000 elementi per volta.
Il modello Gini in Python ha una leggera superiorità sull'implementazione in Prolog per quanto riguarda l'accuratezza e l'errore. Entrambi i modelli hanno una precisione molto bassa e un ricavo basso, indicando difficoltà nel classificare correttamente la classe positiva (pulsar), quindi potenziali problemi con il dataset sbilanciato o con l'approccio stesso del modello.
Il modello Entropia in Python mostra un'accuratezza significativamente migliore rispetto a Prolog. Anche l'errore è molto più basso. Inoltre, Python ha una precisione molto alta ed un buon ricavo, suggerendo una capacità nettamente superiore di identificare correttamente la classe positiva. L'implementazione Prolog conserva gli stessi valori di precisione e ricavo, ma il numero di falsi positivi e falsi negativi è più elevata rispetto a Python. Il modello Entropia in Python supera nettamente l'implementazione Prolog in termini di accuratezza, errore e capacità di classificare correttamente la classe positiva. La discrepanza potrebbe derivare da una ottimizzazione del codice.
In sintesi, il modello Entropia è generalmente più accurato e con un errore minore rispetto al modello Gini, che in entrambe le implementazioni fatica notevolmente con la classe positiva, segnalando una preferenza verso la classe dominante. Il dataset fortemente sbilanciato sembra influenzare negativamente i modelli. Quindi, il modello basato su Entropia è il più adatto per il problema in esame, in particolare con l'implementazione in Python, anche se non viene effettuata la potatura.