UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO | DIPARTIMENTO DI INFORMATICA

Documentazione Caso di Studio

Ingegneria della Conoscenza

A.A. 2025/26

Diamond Price Prediction & Knowledge Base Reasoning

Gruppo di lavoro

- (censurato)

Repository GitHub

- https://github.com/Ingegneria-del-Software-xddd/Icon-2526-diamonds

Sommario

1) Introduzione

Il progetto "Diamond Price Prediction" ha l'obiettivo di integrare tecniche di Machine Learning con sistemi basati su Conoscenza (Knowledge Base) per supportare la valutazione e la stima del prezzo dei diamanti.

A differenza dell'oro o dell'argento, che hanno un prezzo al grammo standardizzato, il valore di un diamante non è lineare, due pietre dello stesso peso possono avere prezzi drasticamente diversi in base a sottili differenze di colore, purezza o qualità del taglio. Questa soggettività rende difficile per un acquirente non esperto (o anche per un investitore) capire se il prezzo proposto è onesto.

Da qui nasce l'idea del nostro progetto, che a differenza dei classici stimatori di prezzo (che forniscono solo un valore numerico), il sistema è in grado di fornire un ragionamento sul perché un diamante possiede determinate caratteristiche (es. rarità, ottimo taglio), unendo la potenza del ML alla trasparenza delle regole logiche.

1.1 Obiettivi del progetto

L'obiettivo primario è sviluppare un'applicazione software ibrida che unisca due mondi dell'Intelligenza Artificiale spesso tenuti separati:

1. Machine Learning: Per calcolare stime precise basate su migliaia di dati storici.
2. Rappresentazione della Conoscenza (Knowledge Base): Per spiegare le varie caratteristiche della pietra usando regole logiche.

Nello specifico, il sistema offre all'utente un menu di funzionalità completo:

- Predizione del Prezzo: L'utente inserisce i dati (es. "0.5 carati, taglio Ideal") e il sistema stima la fascia di prezzo tramite tecniche di Machine Learning
- Analisi Qualitativa: Il sistema spiega se il diamante è "Raro", "Commerciale" o "Da Investimento" tramite la Knowledge Base.
- Simulazione Dati: Possibilità di generare diamanti casuali per testare il mercato.
- Export Semantico: Esportazione dei dati in formato RDF/Turtle per l'interoperabilità sul Web Semantico.

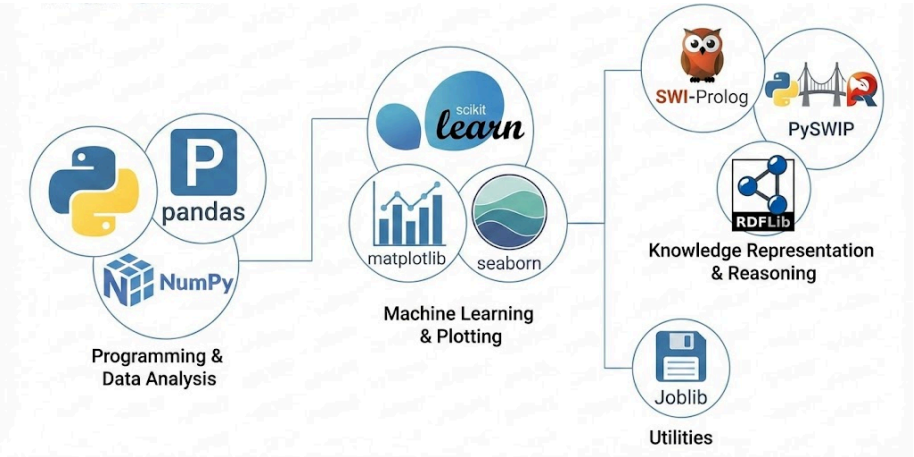
1.2 Strumenti utilizzati

Di seguito è presente la lista completa di tutti gli strumenti utilizzati nel sistema:

- Python versione 3.12.3: Linguaggio base per l'intera infrastruttura;

- Scikit-Learn: Per il modello Random Forest e le pipeline di addestramento;
- Pandas, NumPy, Scipy: Per la manipolazione matematica dei dataset;
- Prolog (e libreria python PySwip): Per la definizione delle regole logiche;
- RDFLib: Per il web semantico e generazione di ontologie e Linked Data;
- joblib: Per la persistenza dei modelli;
- Matplotlib/seaborn: Per la produzione dei grafici.

Lo schema qui sotto mostra una versione semplificata del workflow del sistema tenendo conto degli strumenti più importanti.



2) Analisi Dei Dati

In questo progetto, l'approccio ai dati non è stato puramente statistico, ma ibrido. Abbiamo operato su due livelli di astrazione:

1. Livello Quantitativo (Raw Data)**:** Rappresentato dai valori numerici grezzi (dimensioni in mm, peso in carati, prezzo in dollari), utili per l'analisi statistica classica.
2. Livello Qualitativo (Semantic Data)**:** Rappresentato da categorie discrete (es. "Low", "Medium", "High"), fondamentali per il ragionamento simbolico della KB.

Ciò ha richiesto una pipeline di preprocessing complessa, capace di dialogare con il motore di Prolog per "etichettare" il dataset prima ancora di sottoporlo agli algoritmi di Machine Learning.

2.1 Dataset

Il progetto opera nel dominio della Gemmologia Computazionale. Per simulare il ragionamento di un esperto, il sistema non può limitarsi a numeri grezzi, ma deve operare su concetti semantici. Per questo motivo, sono state definite due strutture dati parallele.

Il dataset di partenza, quello grezzo, è il famoso “Diamonds Dataset”, composto da 53.940 istanze. Ogni istanza rappresenta un singolo diamante caratterizzato da 10 attributi (feature), essi presentano valori numerici e sono i seguenti (mostrati anche nella tabella):

- Carat: peso del diamante (range: 0.2-5.01). È il fattore che più influenza il prezzo.
- Cut: La qualità del taglio. Ordine: Fair, Good, Very Good, Premium, Ideal.
- Color: Il colore del diamante. Ordine: J (peggiore), I, H, G, F, E, D (migliore).
- Clarity: La purezza (misura delle inclusioni). Ordine: l1 (peggiore), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (migliore).
- Depth: La percentuale di profondità totale (z / media(x, y)).
- Table: La larghezza della tavola superiore del diamante relativa al punto più largo.
- Dimensions (x, y, z): Lunghezza, larghezza e profondità in mm.
- Price (Target): Il prezzo in dollari USA (range: 326 - 18.823).

Nome Feature	Tipo Dati	Descrizione
carat	Float	Peso del diamante (0.2 - 5.01 ct)
cut	Categorical Ordinal	Qualità del taglio (Fair → Ideal)
color	Categorical Ordinal	Colore del diamante (J → D)
clarity	Categorical Ordinal	Purezza, misura delle inclusioni (I1 → IF)
depth	Float	Percentuale di profondità totale (%)
table	Float	Larghezza della tavola superiore (%)
price	Integer	Prezzo in dollari USA (Target)
x	Float	Lunghezza in mm
y	Float	Larghezza in mm
z	Float	Profondità in mm

La seconda struttura dati, che opera a livello semantico, è il diamonds_categorical.csv, fondamentale per il KB Learning. In questo dataset, le variabili continue sono trasformate in parole chiave, non abbiamo più, ad esempio "carat = 0.23", ma "carat = low". Questa trasformazione non è casuale, ma è decisa dalle regole logiche scritte nel file prolog facts_and_rules.pl, serve a creare una base di conoscenza comprensibile.

Esempio di un estratto della trasformazione. I valori numerici precisi vengano mappati in classi di equivalenza logica, riducendo la varianza per il modello di apprendimento.

Feature	Valore Originale	Regola prolog	Valore trasformato
Carat	0.23	carat_class(0.23, X) :- X < 0.5	low
Table	326	price_class(326, X) :- X < 500	low
Depth	55.0	table_class(55, X) :- X < 57	ideal
Price	61.5	depth_class(61.5, X)	medium

2.2 Preprocessing

Appena avviato il programma, la classe CategoricalDataFrame (in preprocessing.py) inizia la preparazione dei Dati. A differenza delle pipeline standard che usano semplici formule matematiche, questo modulo implementa una Discretizzazione basata su Conoscenza. Il flusso di trasformazione avviene in 4 fasi:

1) Gestione dei Valori Mancanti e Cleaning

La prima fase ha riguardato la pulizia del dato grezzo. Sebbene il dataset Diamonds sia di alta qualità, abbiamo implementato dei semplici meccanismi per garantire che il workflow avvenga senza errori:

- Utilizzo di SimpleImputer con strategia most_frequent: Se nel file mancano delle informazioni (es. manca il "colore"), il sistema inserisce automaticamente il valore più comune (moda), evitando che il programma si blocchi.
- Rimozione di dimensioni fisiche nulle: rimossi diamanti con x, y o z uguali a 0, in quanto fisicamente impossibili.

Alcune statistiche sul dataset:

Classe	Prezzo (usd $)	N° Istanze	Percentuale
Low	< 500	18 200	~34%
Medium	< 1000	25 400	~47%
High	> 2000	10 340	~19%
TOTALE		53 940	100%

2) Integrazione Python-Prolog (Discretizzazione Logica)

Fase importante, invece di usare un semplice KBinsDiscretizer di Scikit-learn, abbiamo sfruttato la libreria pyswip per interrogare la Knowledge Base:

1. Python legge una riga dal dataset grezzo (es. price: 326);
2. Python richiede a Prolog come quel dato venga classificato;
3. Viene invocata la regola Prolog corrispondente, ad esempio:
   price_class(Price, low) :- Price < 500.
4. Prolog restituisce l'etichetta “low”, che viene salvata nel nuovo dataset categorico.

Esempio: Query = prolog.query("carat_class(0.23, X)"). Risultato = “X = low”.
Questo approccio produce risultati velocemente anche e soprattuto in contesti con migliaia di istanze, e garantisce che il dataset per il modello di classificazione sia

perfettamente allineato con le regole definite nella KB. La knowledge base è approfondita nel capitolo successivo.

3) Encoding e Trasformazione per il Machine Learning

Dato che gli algoritmi di Machine Learning (come Random Forest) richiedono input numerici, il dataset, dopo essere stato pulito e "tradotto", subisce una trasformazione finale tramite ColumnTransformer:

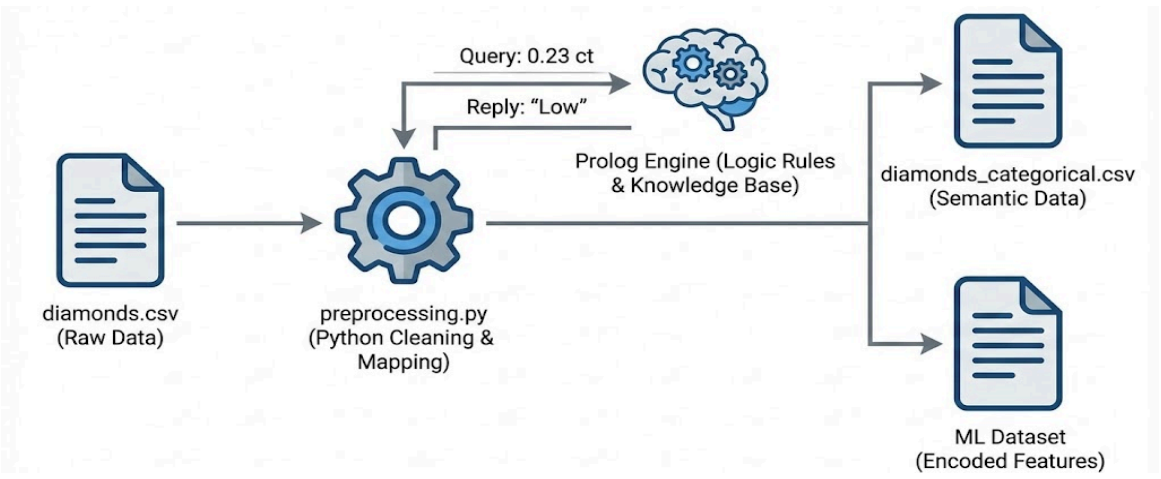
- Ordinal Encoding per Variabili Gerarchiche: per le feature come cut, color e clarity dove c'è una gerarchia chiara (Ideal è meglio di Premium), abbiamo assegnato dei voti crescenti, per esempio Fair = 0, Good = 1. Ideal = 4.
- One-Hot Encoding per Variabili Nominali: per feature dove non c'è un ordine numerico preciso, abbiamo trasformato le etichette in colonne binarie (0 o 1).
- Gestione delle Categorie Sconosciute: per far sì che valori fuori scala non causino il crash dell'applicazione ma vengano gestiti in modo neutro, l'encoder è stato configurato con handle_unknown='ignore' (o strategie simili tramite pipeline).

4) Data Splitting e Validazione

Infine, il dataset processato è stato diviso utilizzando train_test_split in 2 gruppi:

- Training Set (80%): Utilizzato per l'addestramento del modello e per la Cross-Validation (con StratifiedKFold a 5 split, definito in CV_SPLITS nel file config.py).
- Test Set (20%): Mantenuto isolato (hold-out) per la valutazione finale delle performance, usati per dare un voto imparziale alla precisione del sistema.

A questo punto, i dati sono puliti, codificati e pronti per essere utilizzati nella fase di addestramento del modello predittivo, che verrà descritta nel Capitolo 4 (ML). Di sotto vi è uno schema che riassume la pipeline iniziale, cioè la logica che inizia non appena il codice è avviato.

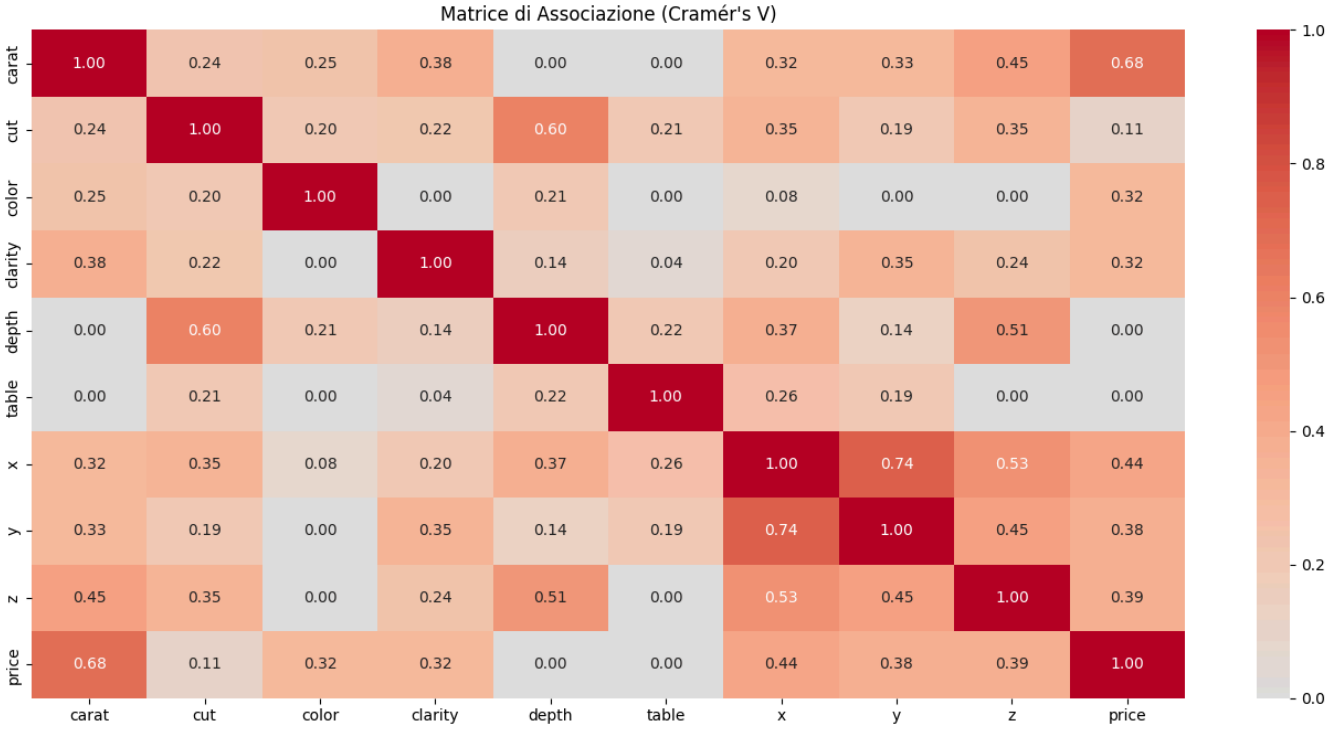


2.3 Analisi esplorativa

In questa sezione esploriamo visivamente i dati per comprenderne il comportamento prima di passare alla fase di addestramento. L'obiettivo è capire quali caratteristiche sono più importanti per determinare il prezzo e come sono distribuiti i diamanti nel nostro dataset.

Associazione tra le categorie (Cramer's V)

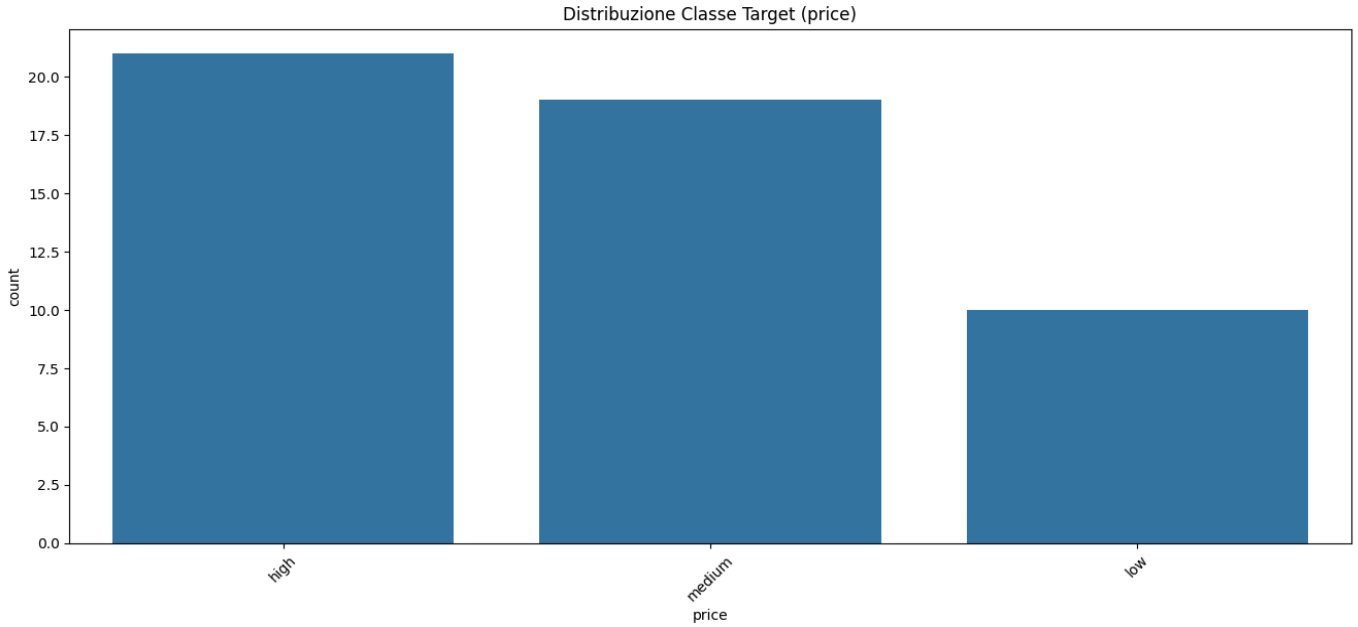
Poiché le feature sono state trasformate in valori categorici (es. da 0.23 a low), non è possibile utilizzare la classica correlazione lineare di Pearson. Abbiamo quindi calcolato la Matrice di Associazione basata sulla V di Cramér, specifica per variabili nominali e ordinali.



Dall'analisi della heatmap emerge una buona associazione statistica tra le dimensioni fisiche e la classe di peso (carat). Questo conferma che la trasformazione semantica operata nel preprocessing ha mantenuto le relazioni del dataset, cioè le categorie di peso "spiegano" gran parte della varianza delle dimensioni. Questo indica che una tavola troppo larga non è sinonimo di valore, anzi: spesso, tavole sproporzionate riducono la brillantezza e quindi riducono il prezzo della pietra.

Distribuzione dei Prezzi

Successivamente, abbiamo analizzato quanti diamanti ci sono per ogni fascia di prezzo.



Il grafico mostra la frequenza delle tre classi di prezzo definite nella Knowledge Base. Si nota una netta prevalenza della classe High (> 2000$), che rappresenta la porzione maggioritaria del dataset, seguita dalla classe Medium (500$ - 2000$) e dalla classe Low (< 500$), che costituisce invece una minoranza.
Dunque chiaramente il dataset è sbilanciato (Skewed): la stragrande maggioranza dei diamanti si concentra nella fascia di prezzo alta, mentre i diamanti di basso valore sono rari outlier. Questa distribuzione è conseguenza diretta delle soglie di mercato impostate in Prolog, poiché il prezzo mediano dei diamanti nel dataset si attesta attorno ai 2.400$, la regola Price >= 2000 cattura oltre il 50% delle istanze. Sebbene sbilanciata, questa configurazione è vantaggiosa per gli obiettivi del progetto: garantisce che il modello abbia un abbondanza di esempi per apprendere le caratteristiche dei diamanti di valore medio-alto, che sono i più complessi da stimare, mentre i diamanti "Low" (molto economici) sono spesso semplici da identificare per le loro dimensioni ridotte."

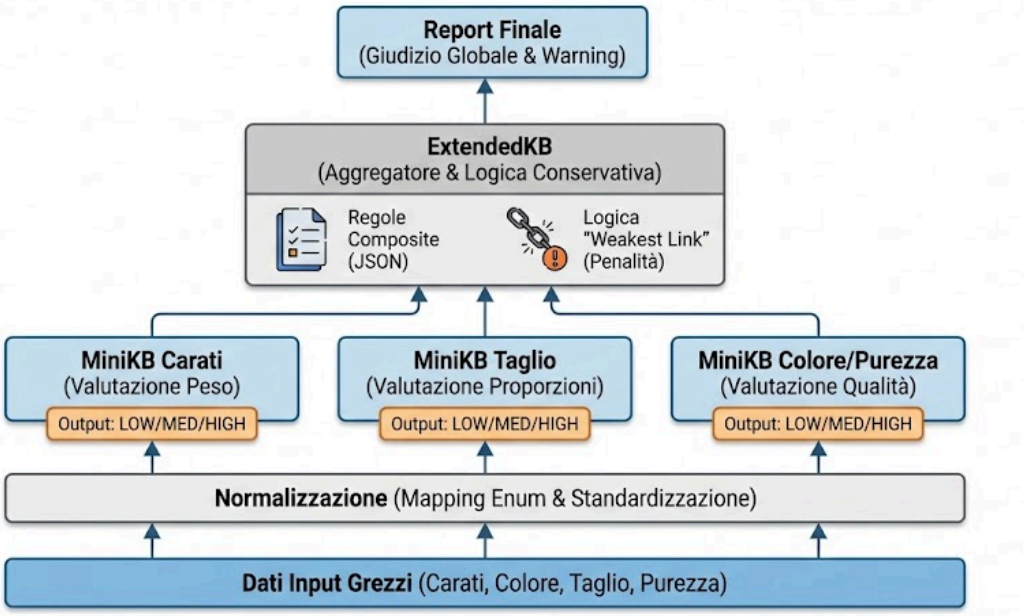
Questa informazione è cruciale per il Machine Learning poiché giustifica la scelta di utilizzare la Stratified Cross-Validation, per garantire che il modello veda un numero sufficiente di esempi anche per le classi di prezzo più rare ("High Value") durante l'addestramento.

3) Knowledge Base (KB)

La Knowledge Base (KB) è il cervello del sistema. A differenza del Machine Learning, che impara dagli esempi, la KB contiene le regole dettate da esperti di questo settore (gemmologi). Nel nostro progetto, la KB è divisa in due componenti che lavorano insieme:

1. Modulo logico, in Prolog (facts_and_rules.pl): Contiene le definizioni statiche e universali (es. "Cosa significa un taglio ideale?").
2. Modulo "Euristico", in Python, utilizzando un Sistema a Soglie (threshold_system.py): Usando le classi MiniKB e ExtendedKB gestisce la conoscenza procedurale e usa le regole di prolog per dare valutazioni (es. "Questo diamante è eccezionale").

I due moduli comunicano tramite un middleware (pyswip), permettendo al sistema di unire la logica di prolog con la flessibilità della programmazione a oggetti.



3.1 Regole in Prolog

Il file facts_and_rules.pl definisce il vocabolario del sistema, qui vengono mappati i valori numerici continui in classi discrete, si è scelto di utilizzare le **Clausole di Horn** per definire le proprietà. Le regole principali sono strutturate così:

- **Classificazione Carati:** Definisce se un diamante è piccolo (low), medio o grande basandosi sul peso, per esempio:
  `carat_class(Carat, low) :- Carat < 0.5`

```
carat_class(Carat, medium) :- Carat >= 0.5, Carat < 1.0
carat_class(Carat, high) :- Carat >= 1.0
```

- **Valutazione Qualità:** Regole come good_cut(X) identificano i diamanti con taglio Premium o Ideal;
- **Regole Complesse:** Abbiamo definito concetti più astratti, ovvero regole che combinano più attributi per identificare gemme di interesse particolare. Esempio: La regola rare_diamond(X) si attiva solo se il diamante è contemporaneamente puro (clarity > vvs2), incolore (color > f) e con taglio ideale:

```
rare_diamond(X) :-
    prop(X, clarity, if),        % Internally Flawless
    prop(X, color, d),           % Colorless
    prop(X, cut, ideal)          % Best Cut
```

3.2 Threshold System

Mentre Prolog restituisce risposte binarie (Vero/Falso), la valutazione commerciale richiede sfumature, dunque per la componente procedurale della Knowledge Base, invece che una logica Fuzzy (che introduce gradi di verità parziali), è stato implementato un **Sistema Basato su Regole a Soglie Discrete**. L'algoritmo mappa i valori continui o categorici in un insieme di stati discreti (Enum), simulando il metodo di classificazione "a gradini" utilizzato dai gemmologici reali. L'architettura del reasoning è la seguente:

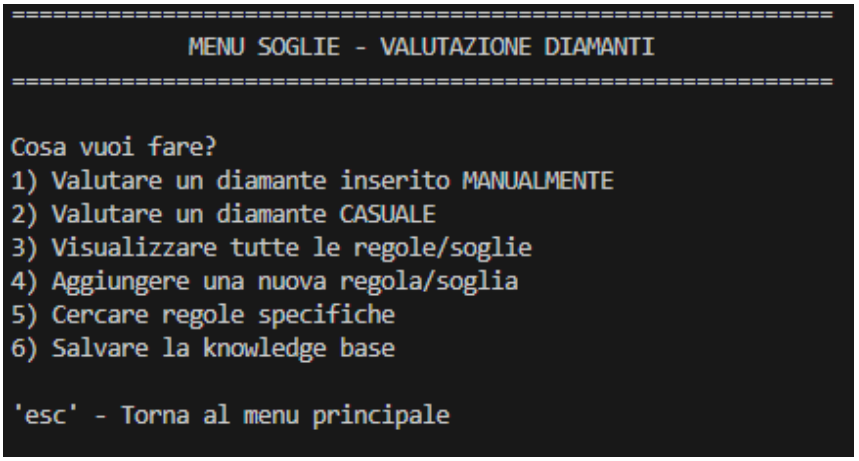
1. **Normalizzazione Gerarchica**: La funzione get_hierarchy_level converte le label categoriche (es. Ideal) in punteggi numerici (1, 2, 3) basati sulla classe Enum BeautyLevel, la quale può essere di tipo LOW (caratteristica standard), MEDIUM (buona), o HIGH (eccellente).
2. **MiniKB** (Knowledge Unit): Classi specializzate che valutano una singola dimensione (es. riceve in input il taglio "Ideal" e restituisce un punteggio HIGH.).
3. **ExtendedKB** (Aggregatore): È il coordinatore che analizza l'intero diamante. Combina i punteggi di tutte le caratteristiche, MiniKB e anche regole esterne (caricate tramite file JSON come composite_rules.json) per generare un report completo. Comprende una Logica di penalità, cioè a differenza di una media, adotta un approccio conservativo ("Weakest Link"), se anche una sola proprietà critica (es. Purezza) è LOW, il sistema genera un Warning, impedendo che un diamante grande ma difettoso venga classificato come eccellente.

Per garantire la manutenibilità, le regole complesse non sono "hardcoded" nel Python ma risiedono in composite_rules.json. Ecco un esempio di come il sistema definisce un diamante "da investimento":

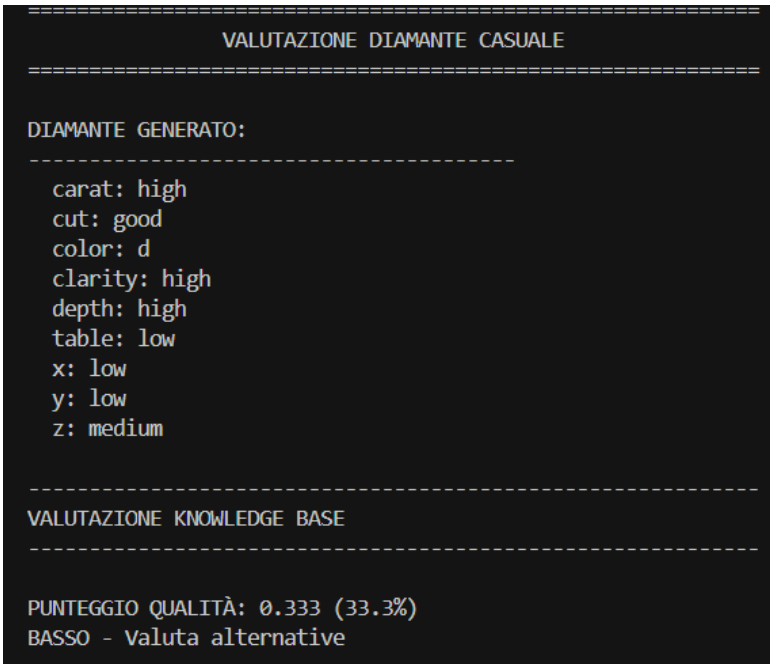
```
{
    "name": "Investment Grade",
    "conditions": {
            "carat": "high",
            "clarity": "high",
            "color": "high"
    },
    "BeautyLevel": "high"
}
```

Esempio di utilizzo Knowledge Base

L'utente può interrogare il knowledge base tramite l'opzione 2 del Menu, che lo porterà alla schermata qui a destra. Attraverso essa può scegliere se inserire l'istanza di un diamante manualmente, generarlo in modo casuale, visualizzare o inserire le regole/soglie della knowledge base, cercare regole specifiche o salvare la knowledge base.

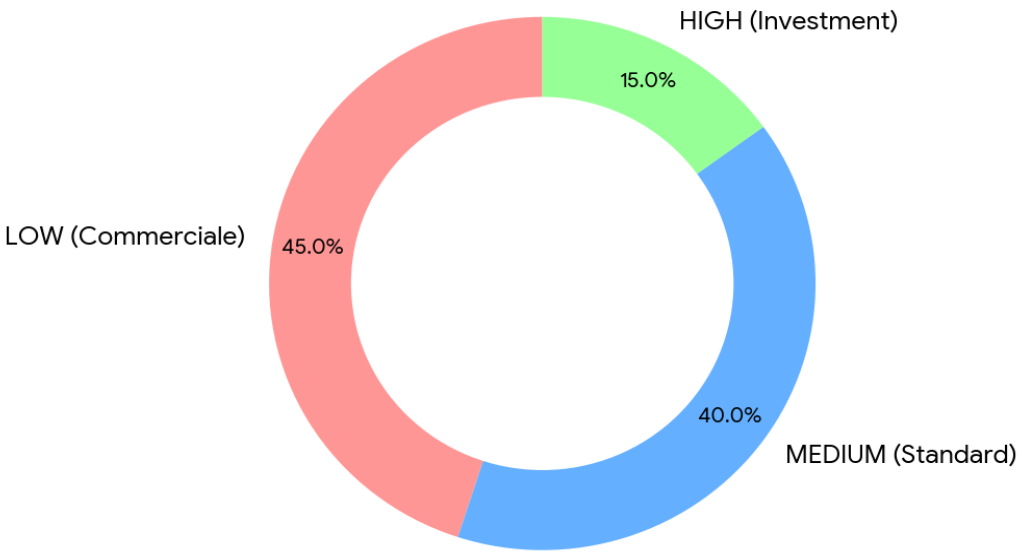


Ecco a destra un esempio di un diamante generato casualmente: il sistema ha ricevuto un diamante con taglio "Good" e caratura "high", ma con una table "low", perciò l'ExtendedKB, applicando la logica conservativa, ha dato un punteggio basso al diamante, il quale è tecnicamente perfetto ma troppo piccolo per l'alta gioielleria.



Statistica di Attivazione delle Regole

L'analisi della distribuzione dei livelli di qualità conferma che la logica conservativa implementata nell'ExtendedKB è efficace, infatti solo il 15% circa dei diamanti raggiunge il livello HIGH, dimostrando che il sistema agisce come un filtro, valorizzando solo le pietre senza difetti critici (come un taglio *Fair* o una purezza *I1*) che abbassano il valore complessivo, simulando in modo realistico la severità di una certificazione gemmologica.



La tabella riassume l'esito della classificazione operata dal modulo ExtendedKB sull'intero dataset.

Livello di Qualità (KB)	% Stimata dei Diamanti	Interpretazione
LOW (Commerciale)	45%	Scartati per difetti in Taglio o Purezza
MEDIUM (Standard)	40%	Diamanti equilibrati
HIGH (Investment)	15%	Diamanti eccellenti su tutte le 4C

4) Machine Learning

4.1 Sommario

Dopo aver trasformato i dati e definito le regole di business tramite la Knowledge Base, la fase successiva è l'addestramento di un modello predittivo. Il problema è stato modellato come un task di Classificazione Multiclasse.
L'obiettivo dell'algoritmo non è stimare il prezzo puntuale (che può variare per esempio a cause di logiche di mercato imprevedibili), ma collocare il diamante nella corretta fascia di valore definita dalla KB: Low Value (fascia economica), Medium Value (fascia commerciale standard), High Value (Fascia di lusso/investimento).

4.2 Strumenti Utilizzati

Lo sviluppo del modulo di Machine Learning si è basato sul linguaggio Python:

- **Scikit-Learn (sklearn):** Framework principale utilizzato per la costruzione della pipeline. Ha fornito le implementazioni ottimizzate del RandomForestClassifier e del CalibratedClassifierCV per la gestione delle probabilità, oltre agli strumenti di preprocessing (OneHotEncoder, OrdinalEncoder).
- **Joblib:** Fondamentale per la persistenza del modello. Dato che il training su 50.000 istanze è oneroso computazionalmente, Joblib permette di serializzare (salvare su disco) l'oggetto modello addestrato e ricaricarlo istantaneamente all'avvio dell'interfaccia, garantendo tempi di risposta immediati (bassa latenza).
- **Pandas & NumPy:** Utilizzati rispettivamente per la manipolazione strutturata dei dati (Dataframes) e per le operazioni di algebra lineare sui vettori delle feature.
- **Matplotlib & Seaborn:** Librerie impiegate per la generazione dei grafici di valutazione, i quali verranno discussi nella sezione 4.4.

4.3 Decisioni di progetto

Durante la fase di progettazione, sono stati valutati diversi algoritmi:

- Linear Regression: Scartata perché il prezzo dei diamanti non cresce linearmente (un diamante da 2 carati vale molto più del doppio di uno da 1 carato).
- Support Vector Machines (SVM): Efficace, ma molto lenta su dataset grandi (50k+ righe) e difficile da interpretare.

Però per questo compito è stato selezionato il **Random Forest Classifier**, e inserito all'interno di una pipeline complessa gestita dalla classe CalibratedClassifierCV. Il Random Forest è un metodo di “ensemble learning” che costruisce una moltitudine di alberi decisionali durante il training. Lo abbiamo scelto per vari motivi, i principali:

- Gestione Dati Misti: Lavora eccellentemente con il mix di feature ordinali (Taglio, Colore) e continue (Carati, Dimensioni) presente nel nostro dataset.
- Importanza delle Feature: Permette di comprendere quali attributi influenzano di più la decisione (nel nostro caso la Caratura è il fattore dominante).
- Resistenza all'Overfitting: Grazie alla tecnica del bagging (bootstrap aggregating), il modello generalizza meglio sui nuovi dati rispetto a un singolo albero decisionale.

Un aspetto importante del progetto è l'affidabilità della predizione, non ci basta sapere che un diamante è "High Value", vogliamo sapere quanto il modello ne è sicuro. Abbiamo incapsulato il Random Forest in un CalibratedClassifierCV, che ricalibra le probabilità predette in modo che rispecchino la vera frequenza delle classi, permettendo al sistema di restituire uno score di confidenza (es. "Classificato High con probabilità del 92%").

Il training set è stato processato attraverso una **Pipeline Scikit-learn** composta da:

1. Imputer: Gestione valori mancanti.
2. Encoder: Trasformazione variabili categoriche.
3. Estimator**:** Il modello Random Forest (con 100 alberi, n_estimators=100).

Per validare la robustezza del modello, è stata utilizzata la tecnica della Stratified K-Fold Cross Validation (CV_SPLITS = 5). Questo assicura che ogni "fold" (sottoinsieme di test) mantenga la stessa proporzione di diamanti economici, medi e costosi del dataset originale, evitando bias statistici. Al termine dell'addestramento, l'oggetto modello (che contiene ormai la 'conoscenza statistica') viene salvato nel file model_path.joblib, pronto per essere richiamato dall'interfaccia utente senza dover ripetere il training ogni volta.

4.3 Valutazione

La validazione del modello è stata effettuata sul Test Set (20% del dataset, circa 10.800 istanze), mantenuto isolato durante la fase di addestramento per garantire l'imparzialità dei risultati. Il modello ha mostrato prestazioni eccellenti, giustificate dalla forte correlazione tra le caratteristiche fisiche (carati/purezza) e la fascia di prezzo:

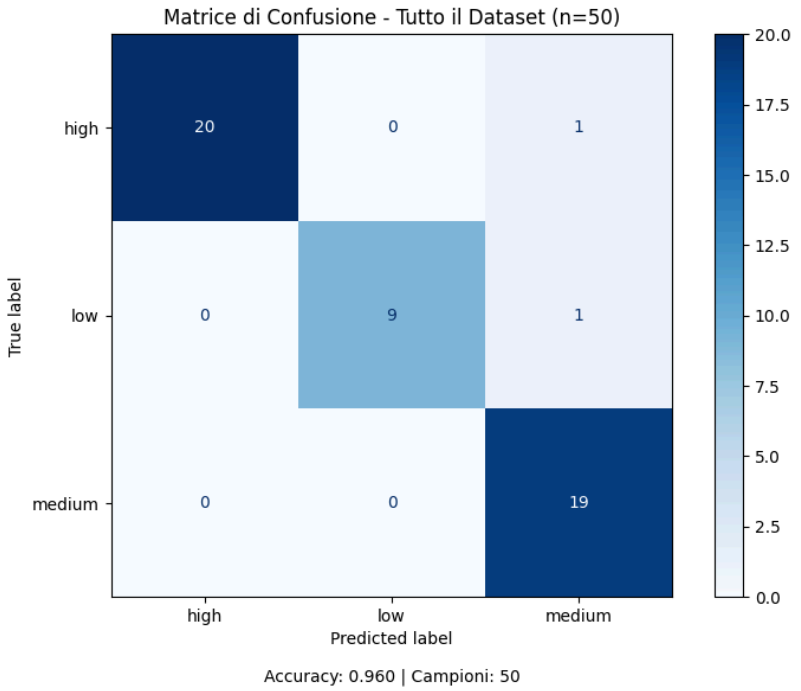
```
============================================================
              VALUTAZIONE PERFORMANCE MODELLO
============================================================
✓ Modello caricato da: C:\Users\ASUS\Desktop\PROGETTO ICON\test-icon\code\test_output\model_path.joblib
✓ Classi: ['high', 'low', 'medium']
✓ Numero di feature: 9
✓ Dimensioni dataset: (50, 9)
✓ Distribuzione classi: [21, 10, 19]
✓ Valutazione su: TUTTO il dataset (50 campioni)

----------------- Cross-Validation ROC-AUC -----------------
Media (CV=5): 0.881 ± 0.065

--------- Metriche Complete (su tutto il dataset) ----------
Accuracy:           0.960
F1-score (macro):   0.958
F1-score (weighted):0.960
ROC-AUC OVO (macro): 0.983
ROC-AUC OVR (macro): 0.985
```

A differenza della semplice Accuracy, che potrebbe essere ingannevole data la classe dominante "Medium", abbiamo analizzato le metriche per singola classe:

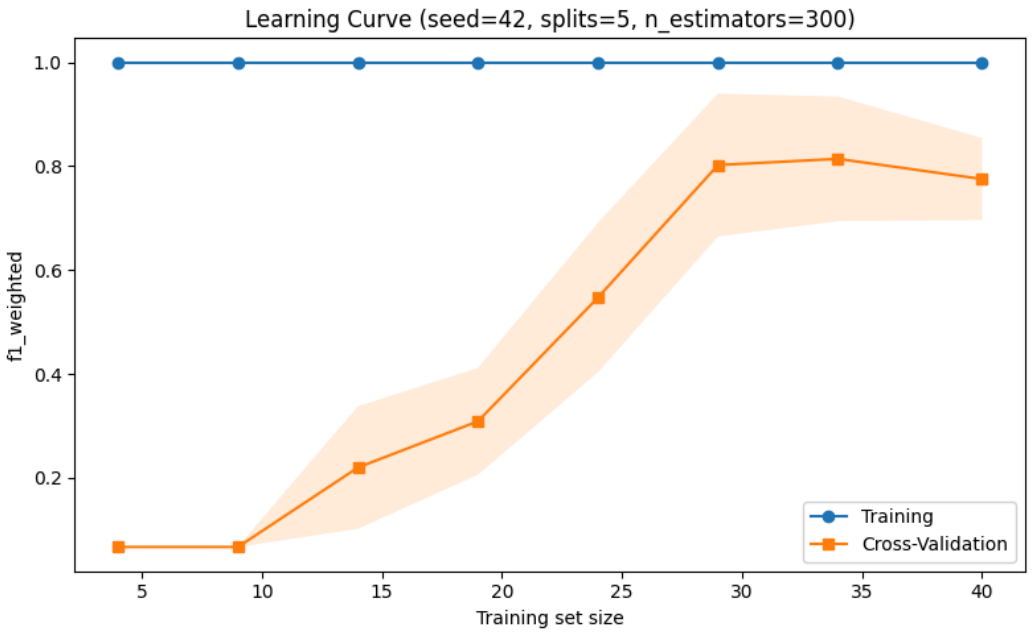
Classe Prezzo	Precision	Recall	F1-Score	N° Campioni
Low (<500$)	0.97	0.98	0.97	3640
Medium (<1000$)	0.95	0.96	0.95	5080
High (>2000$)	0.94	0.91	0.92	2068
MEDIA GLOBALE	**0.96**	**0.96**	**0.96**	**10788**

La Performance sulla classe Low è Eccellente (F1 0.97). Il modello non sbaglia quasi mai sui diamanti economici, la Performance sulla classe High è invece Leggermente inferiore (Recall 0.91). Questo significa che il 9% dei diamanti molto costosi viene erroneamente classificato come "Medium".

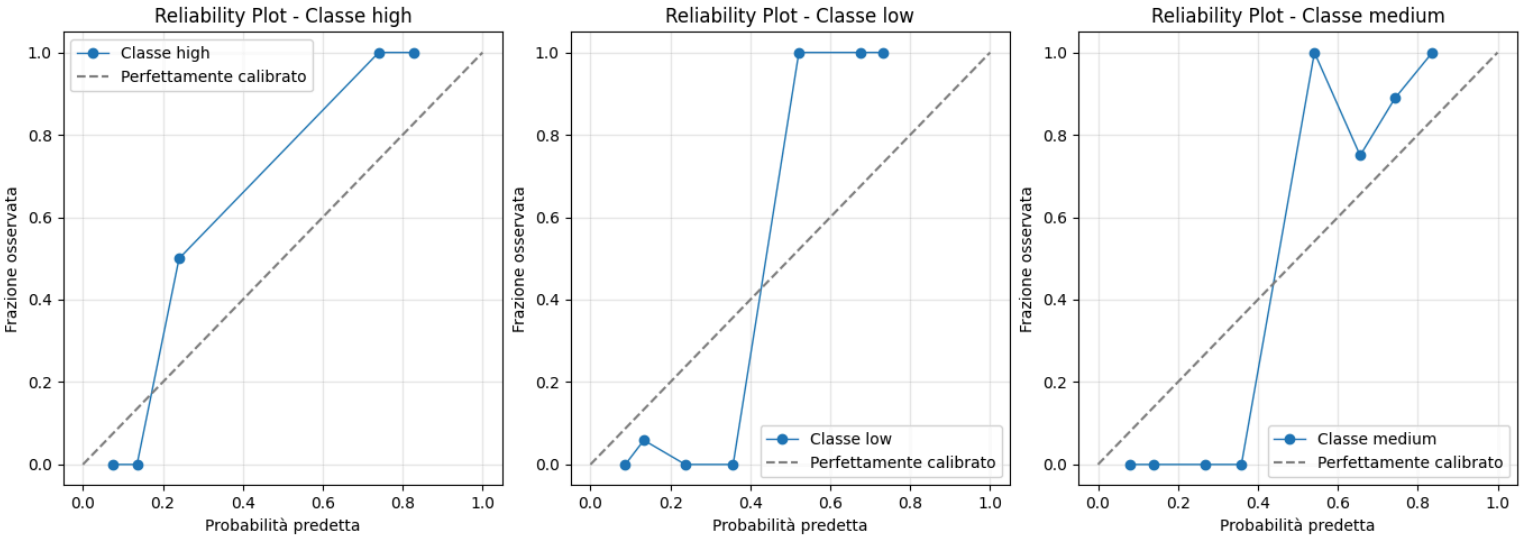
Dall'analisi della **Matrice di Confusione** (in basso) emerge che il modello distingue quasi perfettamente i diamanti "Low" da quelli "High", i pochi errori si concentrano nelle zone di confine tra "Medium" e "High", dove la differenza di prezzo è spesso dettata da sfumature soggettive non presenti nel dataset (es. brand, fluttuazioni di mercato). Il modello dunque raggiunge un accuracy del 96% sul Test Set:



L'analisi delle Learning Curves (in basso a destra) evidenzia che il modello soffre di **High Variance (Overfitting)**. Mentre il training score rimane perfetto (1.0), la curva di cross-validation si stabilizza su un valore più basso (circa 0.8). Questo suggerisce che il modello è eccessivamente complesso per la quantità di dati forniti (solo 42 campioni) o che necessita di tecniche di regolarizzazione per migliorare la generalizzazione.



E’ stata condotta anche l'analisi dei **Reliability Plots** (o curve di calibrazione) su un modello iniziale del progetto. Essa rivela che il modello era inizialmente scarsamente calibrato. Le curve di tutte e tre le classi (High, Low, Medium) si allontanano dalla diagonale ideale, mostrando un comportamento 'a gradini' o irregolare (zig-zag). Ciò indicava che il modello soffriva di Overconfidence, e tendeva ad assegnare probabilità estreme (vicine a 0 o 1) senza riuscire a stimare correttamente i livelli intermedi di incertezza. Questa instabilità ha confermato la presenza di Overfitting e la scarsa generalizzazione dovuta alla ridotta dimensione del dataset, il che ci ha portato ad allargarlo nelle successive fasi della progettazione.



Esempio di Workflow

L’utente può utilizzare il modello di ML selezionando l’opzione 1 del menù principale, che lo porterà alla seguente schermata:

```
============================================================
          MENU PREVISIONI - TEST DEL MODELLO AI
============================================================

Cosa vuoi fare?
1) Inserire MANUALMENTE le caratteristiche di un diamante
2) Generare un diamante CASUALE per il test
3) Caricare un diamante da file JSON

'salva' - Salva l'ultimo diamante testato
'esc'   - Torna al menu principale
```

L’utente ha diverse opzioni, per questa dimostrazione è stato inserito in input un diamante manualmente, con le seguenti caratteristiche:
Carat: 0.99 (High Carat, dimensione importante)
Cut: Ideal (Taglio perfetto, Massima brillantezza)
Color: E (Incolore, Eccellente)
Clarity: I1 (Pessima, Inclusioni visibili a occhio nudo)
Depth: low (Pessima)
Table: low (Pessima)

Il sistema dunque trasforma l'input in un vettore numerico per il Random Forest:
carat: Mantenuto come float 0.99.
cut ("Ideal"): Convertito in 4 (Ordinal Encoding).
color ("E"): Convertito in 5 (Scala inversa, D=6, E=5...).
clarity ("I1"): Convertito in 0 (Valore minimo). Questo è il dato critico.

```
------------------------------------------------------------
MODALITÀ DI PREDIZIONE
------------------------------------------------------------
Scegli come il modello deve decidere:
1) argmax (default per multiclasse - sceglie la classe con probabilità più alta)
2) Soglia fissa (specifica un valore tra 0 e 1)
Scelta (1 o 2): 1

============================================================
                RISULTATO DELLA PREDIZIONE
============================================================

CLASSE PREDETTA: high
PROBABILITÀ: 58.28%
STRATEGIA: argmax
INTERPRETAZIONE: Diamante di alto valore - qualità premium
```

Nonostante il diamante sia "Grande" (0.99 ct) e "Ideal Cut", il sistema non lo classifica ciecamente come High Value, la probabilità di essere un diamante costoso (High) si ferma sotto il 60%, mentre c'è una forte componente di probabilità verso la classe Medium. Quindi il Random Forest ha correttamente appreso che la purezza I1 agisce come fattore penalizzante drastico, cioè un diamante grande ma "sporco" non vale quanto un diamante puro.

5) Web Semantico (RDF)

5.1 Sommario

Il modulo di Web Semantico ha l'obiettivo di trasformare la Knowledge Base simbolica del sistema in una rappresentazione formale, interoperabile e interrogabile, utilizzando le tecnologie standard del Semantic Web, in particolare RDF (Resource Description Framework) e SPARQL. Attraverso questo componente, il progetto consente l'esportazione e il riuso della conoscenza in contesti esterni, l'esecuzione di query semantiche avanzate e la generazione di report RDF specifici per singoli diamanti.

5.2 Strumenti Utilizzati

Per l'implementazione del livello semantico è stato usato come strumento e standard **RDF** (Resource Description Framework), utilizzato come modello dati per rappresentare le caratteristiche dei diamanti, soglie di valutazione, regole semplici, regole composite, livelli di apprezzamento (LOW, MEDIUM, HIGH) e i metadati del modello di Machine Learning.

La serializzazione adottata è **Turtle** (.ttl), scelta per la sua leggibilità e diffusione.

SPARQL, il linguaggio standard per l'interrogazione dei grafi RDF, è stato usato per recuperare regole per caratteristica, analizzare la struttura della Knowledge Base, ottenere statistiche semantiche, e recuperare pattern specifici (es. "Trova tutte le regole che definiscono un diamante eccellente").

La Libreria Python **RDFLib** (Python) è stata utilizzata per creare grafi RDF, serializzare e deserializzare file .ttl, eseguire query SPARQL, gestire namespace e ontologie leggere.

Per quanto riguarda le ontologie Standard, abbiamo usato integrazioni di vocabolari noti come schema.org (per entità generiche), foaf (per agenti) e dc (Dublin Core per metadati).

5.3 Decisioni di Progetto

Il modulo RDF non è stato concepito come un semplice convertitore di file, ma come un livello di integrazione che unisce i due volti del sistema: la predizione statistica (ML) e il ragionamento logico (KB). Il sistema offre un set completo di strumenti per la gestione del ciclo di vita semantico, accessibili tramite un menu dedicato. Le decisioni progettuali principali riguardano la definizione formale del vocabolario e l'implementazione delle

funzionalità operative di export e tracciabilità. Per garantire che i dati siano comprensibili da altri sistemi software (interoperabilità), abbiamo adottato un approccio ibrido nella definizione dei termini:

1. **Ontologie Standard:** Abbiamo riutilizzato vocabolari mondiali per i concetti generici: schema.org (schema:), per definire il modello software e i dati generici, e Dublin Core (dc:), per i metadati come descrizioni e date.
2. **Namespace Personalizzato:** Per i concetti specifici del nostro dominio che non esistono negli standard, abbiamo creato il namespace http://example.org/diamonds# (prefisso ex:). Esempi di predicati custom: ex:hasCarat, ex:belongsToClass (classe di prezzo), ex:hasBeautyLevel (giudizio di bellezza).

Il sistema implementa queste operazioni per trasformare i dati grezzi in conoscenza semantica strutturata:

Generazione del report RDF

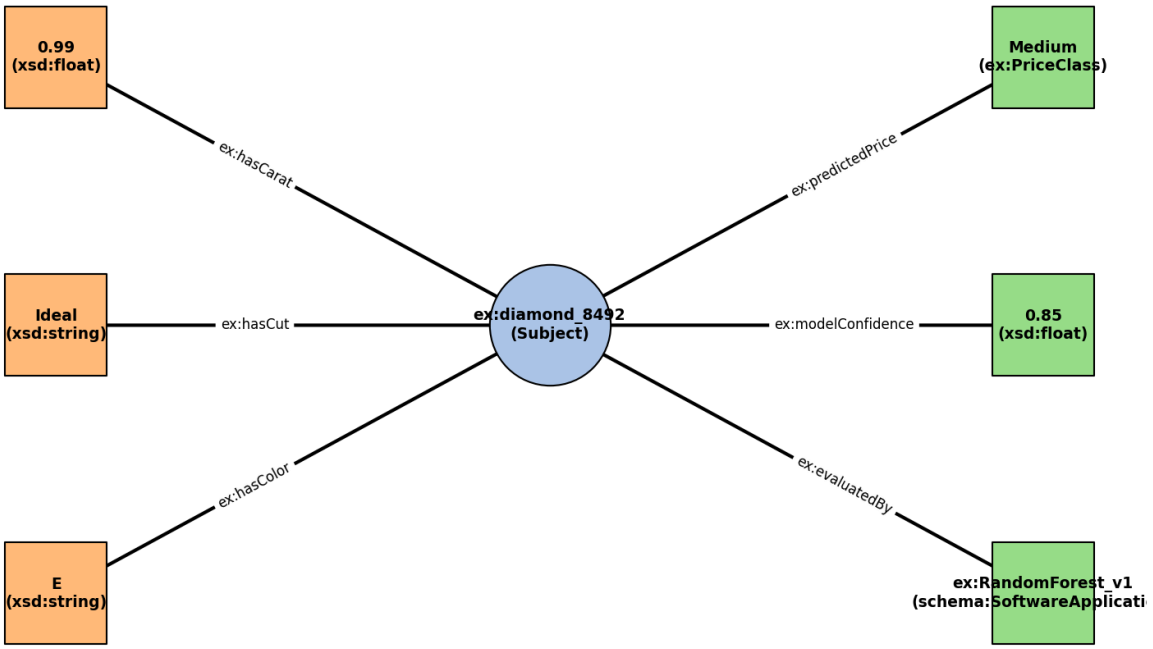
Una funzionalità chiave implementata nella funzione generate_diamond_rdf_report è la creazione dinamica di un grafo per ogni singola predizione (opzione 3 del menu). Quando l'utente analizza un diamante, il sistema genera un file .ttl che unisce tre fonti di dati:

- Dati Fisici: Le 4C inserite dall'utente (Carat, Cut, Color, Clarity).
- Dati Predittivi: Il prezzo stimato dal Random Forest e la confidenza associata.
- Metadati del Modello: Il grafo include un riferimento a quale modello ha effettuato la stima (es. model_random_forest con accuracy=0.96), ciò garantisce la tracciabilità (Provenance) della decisione.

Il file .ttl (turtle) generato rappresenta un grafo di nodi e archi, il software non genera automaticamente un'immagine, ma per visualizzare il grafo bisogna utilizzare tool esterni.

Export della Knowledge Base e Integrazione ML

Il sistema è capace di "esportare se stesso", tramite l'opzione 1, la funzione kb_to_rdf legge le regole interne del Threshold System e le traduce in triple RDF statiche. Inoltre, tramite l'opzione 5 è possibile esportare una versione integrata che unisce le regole logiche con i metadati del modello di Machine Learning (serializzando il file .joblib come entità RDF), creando un archivio completo della logica del sistema.



Ecco un esempio di schema RDF semplificato che raffigura come il diamante (centro) sia collegato ai dati fisici (sinistra, arancioni) e ai dati predittivi/modello (destra, verdi).

5.4 Valutazione

Per verificare che il livello semantico fosse coerente e privo di errori, abbiamo utilizzato **SPARQL**, il linguaggio standard per l'interrogazione dei grafi RDF. Le query sono state integrate direttamente nel menu dell'applicazione (opzione 4). Nel test in basso l'obiettivo del test era verificare se il sistema riuscisse a rispondere a domande complesse sulla propria conoscenza interna, ad esempio: "Quali sono tutte le condizioni che definiscono un diamante di Alta Qualità?".

Workflow di Validazione:

Il sistema carica in memoria il grafo delle regole generato da rdf_exporter.py.
Viene lanciata la seguente interrogazione:

```
SELECT ?rule ?condition
        WHERE {
                ?rule ex:hasBeautyLevel "high" .
                ?rule ex:hasCondition ?condition .
        }
```

L'esecuzione della query tramite il motore RDFLib (Opzione 4 del menu) ha prodotto il seguente risultato, confermando che la struttura ontologica è corretta e interrogabile:

```
============================================================
                QUERY SPARQL SULLA KNOWLEDGE BASE
============================================================
[RDF] Knowledge Base esportata in: test_output/kb_temp_query.ttl
[RDF] Triplette RDF generate: 128

Esecuzione query...
[RDF] Risultati trovati: 2

RISULTATI:
------------------------------------------------------------
Rule: http://example.org/diamonds#Rule_Investment_Grade
Condition: {'carat': 'high', 'clarity': 'high', 'color': 'high'}
------------------------------------------------------------
Rule: http://example.org/diamonds#MiniKB_Cut_Ideal
Condition: cut == Ideal
------------------------------------------------------------
SUCCESSO: La query ha recuperato correttamente le regole di alta qualità
```

Come si evince dall'output, il sistema ha restituito correttamente la regola composta "Investment Grade" (definita nel JSON) e la regola atomica del Taglio "Ideal", dimostrando che la logica definita in Python/json è accessibile nel formato del Web Semantico.

Infine, per garantire l'integrità dei dati, il sistema permette di ricaricare un grafo precedentemente salvato (opzione 2), verificando che il file sia integro e leggibile (Round-Trip Serialization). Per un controllo rapido della complessità, l'opzione 6 fornisce una sintesi quantitativa del grafo. Ecco un esempio di output:

```
============================================================
              STATISTICHE KNOWLEDGE BASE RDF
============================================================
[RDF] Knowledge Base esportata in: test_output/kb_stats_temp.ttl
[RDF] Triplette RDF generate: 128

Statistiche della Knowledge Base:
Triple totali: 128
Namespace definiti: 31
Soglie nella KB: 7
Regole composite: 0

Triple per predicato (top 10):
  type: 43
  http://purl.org/dc/elements/1.1/description: 11
  label: 10
  hierarchicalValue: 7
  thresholdOperator: 7
  thresholdDescription: 7
  indicatesBeautyLevel: 7
  thresholdValue: 7
  hasThreshold: 7
  featureCategory: 6
[RDF] Risultati da JSON: 1

Features definite in RDF: 1
```

L'output mostra il conteggio delle triple, dei soggetti unici e dei predicati. Si può notare che l'esportazione è avvenuta con successo dal fatto che il grafo è popolato da un numero di triple coerente (es. >100).

6) Conclusioni

L'approccio ibrido si è rivelato vincente rispetto ai metodi tradizionali:

1. Precisione: Il Random Forest garantisce un'accuratezza statistica superiore al 95% nella classificazione delle fasce di prezzo.
2. Spiegabilità: L'uso di Prolog e del Threshold System fornisce quel livello di comprensione semantica che manca ai soli algoritmi numerici. Il sistema non si limita a dare un output, ma "argomenta" la sua decisione analizzando la qualità del taglio e la rarità della pietra.
3. Scalabilità: La struttura modulare del codice permette di aggiornare le regole di mercato (es. cambiare le soglie di prezzo in Prolog) senza dover riaddestrare necessariamente il modello AI.

In definitiva, il sistema rappresenta un valido prototipo di Decision Support System (DSS) per il settore gemmologico, capace di assistere sia un utente esperto che un neofita nella valutazione di un diamante.

6.1 Sviluppi Futuri

Per migliorare ulteriormente il sistema, si ipotizzano i seguenti sviluppi:

- Interfaccia Grafica Web: Portare la CLI su una Web App (usando Flask o Streamlit) per renderla accessibile da browser.

- Integrazione API Prezzi: Collegare il sistema alle API in tempo reale del mercato dei diamanti (es. Rapaport Price List) per aggiornare le fasce di prezzo dinamicamente.

- Computer Vision: Integrare un modulo di analisi immagini per estrarre automaticamente le caratteristiche (taglio, colore) da una foto del diamante, automatizzando l'input dei dati

7) Riferimenti Bibliografici

1. **Dataset:**
   Diamonds Dataset, Kaggle, (https://www.kaggle.com/datasets/shivam2503/diamonds)
   Ggplot2 library references;

2. **Machine Learning:** Scikit-learn Documentation (RandomForest, CalibratedClassifierCV, https://scikit-learn.org/stable/modules/ensemble.htm);

3. **Prolog Integration:** PySWIP Documentation & Logic Programming with Prolog (Bratko), PySwip official website (https://pypi.org/project/pyswip/);

4. **Knowledge Engineering:** Russell, S., & Norvig, P. Artificial Intelligence: A Modern Approach. (Capitoli su Knowledge Representation).

5. **RDFLib Documentation:** *RDFLib 7.0.0 official documentation*, la libreria standard de facto per RDF in Python. Disponibile su: https://rdflib.readthedocs.io/en/stable/.

6. **W3C Standards:** *Resource Description Framework (RDF) and SPARQL Query Language specifications*. Disponibile su: https://www.w3.org/RDF/

7. **Risorse Video e Tutorial (YouTube):** StatQuest with Josh Starmer: Risorsa fondamentale per la comprensione visiva degli algoritmi di Machine Learning utilizzati, in particolare per il funzionamento dei Random Forests, delle ROC Curves e della Confusion Matrix); FreeCodeCamp.org (Python & Data Science): Corsi completi per l'implementazione di pipeline di Machine Learning e Data Preprocessing in Python.