Documentazione Completa del Progetto SmartFoodSelector

Gianluca Lascaro (matricola 75867)

3 febbraio 2025

Sommario

Il progetto **SmartFoodSelector** ha come obiettivo la classificazione e la caratterizzazione di prodotti alimentari attraverso l'uso combinato di tecniche di *clustering*, *apprendimento supervisionato*, *reti bayesiane* (in forma continua e discreta) e una *base di conoscenza Prolog* per effettuare inferenze logiche su dati nutrizionali.

Nel presente documento si illustrano in dettaglio la struttura del progetto, le metodologie impiegate, le fasi di elaborazione dei dati e i risultati ottenuti, fornendo una visione completa e approfondita dell'architettura e delle funzionalità di **SmartFoodSelector**.

Indice

1 Problem Definition (Identificazione degli obiettivi)

1.1 Obiettivi del progetto

Lo scopo principale di **SmartFoodSelector** è creare un sistema *integrato* per:

- **Preprocessare** i dati nutrizionali di prodotti (energy, fat, sugar, protein, ecc.);
- **Clusterizzare** i prodotti in gruppi omogenei in base a caratteristiche nutrizionali comuni;
- **Addestrare classificatori supervisionati** per assegnare in modo rapido un nuovo prodotto al cluster corretto;

- **Costruire Reti Bayesiane** (continue o discretizzate) per inferire probabilità condizionali anche con dati mancanti;
- **Fornire una base di conoscenza Prolog** per eseguire query logiche sui prodotti.

Il progetto integra *data science* e *intelligenza artificiale simbolica* (Prolog), fornendo strumenti di supporto decisionale per la nutrizione.

2 Loading the Data and Packages (Caricamento dati e librerie)

2.1 Struttura e Moduli Principali

Il progetto è organizzato in pacchetti e file Python, ciascuno focalizzato su una parte del flusso:

- `src/dataset_preprocessing.py`: Caricamento e preprocessamento (**pulizia**, selezione feature, normalizzazione).
- `src/unsupervised_clustering.py`: Implementa *k-Means* e produce grafici come la "elbow curve" e la distribuzione dei cluster.
- `src/supervised_trainer.py`: Training e valutazione di *Decision Tree*, *Random Forest*, *Logistic Regression*.
- `src/bayes_net.py`: Creazione di Reti Bayesiane (continue/discrete) con *pgmpy*.
- `src/prolog_interface.py`: Integrazione e query di Prolog (`pyswip`).
- `src/generate_prolog_knowledge_base.py`: Generazione della knowledge base Prolog (`knowledge_base.pl`).
- `main.py`: *Script principale* per il coordinamento di tutto il flusso.

Le librerie chiave includono:

- **pandas, numpy, scikit-learn, imbalanced-learn** per data handling e manipolazione;
- **kneed** per la determinazione del gomito (elbow) nel k-Means;
- **optuna** per il tuning di iperparametri (es. Random Forest);
- **shap** per spiegabilità dei modelli;
- **pgmpy** per la costruzione di Reti Bayesiane;
- **pyswip** per interfacciarsi con Prolog.

2.2 Caricamento del Dataset

Il dataset principale (`en.openfoodfacts.org.products.tsv`) è tipicamente collocato in `data/raw`. Viene letto in un `DataFrame pandas` specificando il separatore “ e alcuni accorgimenti per file di grandi dimensioni (`low_memory=False`).

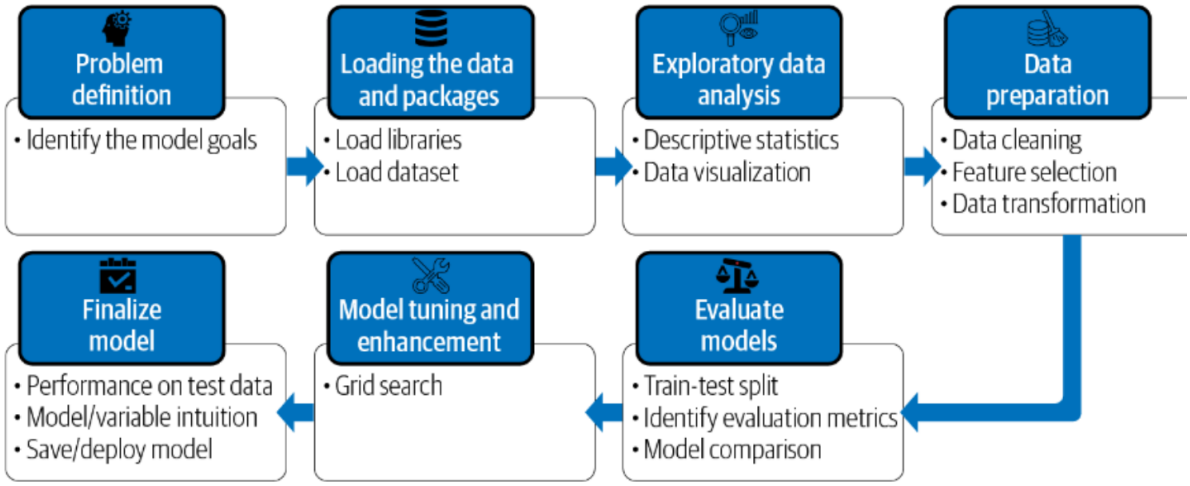

Figura 1: Struttura e fasi di sviluppo del progetto (*Model development steps*).

3 Exploratory Data Analysis (Analisi esplorativa)

3.1 Descriptive Statistics

Dopo il caricamento, è buona norma esplorare i dati per:

- **Valori min/max**, medie e deviazioni standard sulle feature nutritive (`energy_100g`, `fat_100g`, ecc.);
- **Presenza di valori mancanti** e outlier.

3.2 Data Visualization

Si possono usare grafici (es. *histogrammi*, *boxplot*) per investigare la distribuzione delle variabili e individuare eventuali anomalie. Tali passi consentono un *preprocessing* più mirato (es. rimozione righe inconsistenti).

4 Data Preparation (Preprocessing dei Dati)

4.1 Caricamento e Pulizia

Nel progetto **SmartFoodSelector**, i dati del file `en.openfoodfacts.org.products.tsv` vengono caricati in un `DataFrame` e si effettua:

- **Selezione** delle colonne essenziali (`energy_100g`, `fat_100g`, `sugars_100g`, ecc.);
- **Rimozione** di valori `NaN` o nulli;

- (Eventuale) **gestione outlier**, qualora si riscontrino valori errati o troppo estremi.

4.2 Normalizzazione con MinMaxScaler

Per omogeneizzare la scala dei dati, si usa il **MinMaxScaler** (range [0,1]). Le variabili come `energy_100g` e `salt_100g` vengono così poste nello stesso intervallo, riducendo problemi di bilanciamento in fase di clustering.

4.3 Salvataggio in CSV

I dati pre-processati vengono infine salvati (`Dataset.csv`), pronti per le successive fasi di *clustering* e *training*.

5 Evaluate Models (Clustering e Classificatori)

In questa fase si passa all'analisi vera e propria, dove:

1. Prima si individuano gruppi di prodotti simili (*clustering non supervisionato*);
2. Poi si addestrano modelli *supervisionati* per classificare direttamente i nuovi prodotti nei cluster trovati.

5.1 Clustering: k-Means

5.1.1 Elbow Plot

Per determinare il numero di cluster k, si calcola l'*inertia* per diversi valori di k. Si utilizza il "metodo del gomito" (*Elbow Method*), visualizzato tramite la libreria `kneed`.

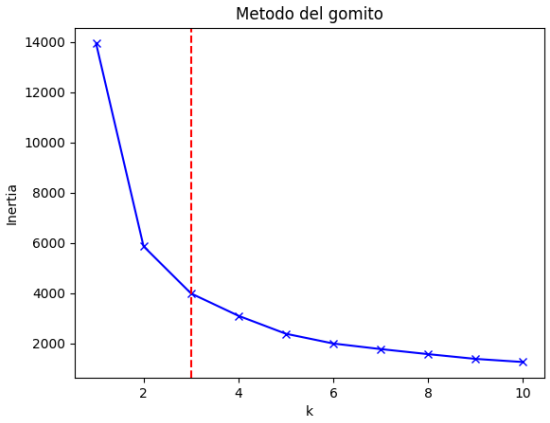

Figura 2: Esempio di *Elbow Plot* che suggerisce $k = 3$.

5.1.2 Assegnazione e Distribuzione dei Cluster

Confermato $k = 3$, si applica k-Means e si aggiunge una colonna `cluster` nel dataset. Il risultato (ad es. `clustered_dataset.csv`) contiene l'etichetta di cluster per ogni prodotto.

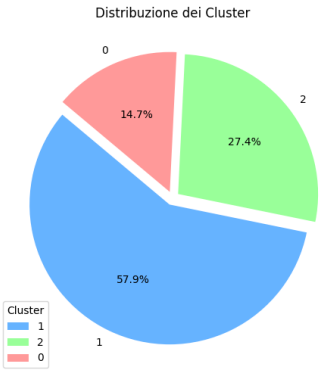

Figura 3: Grafico a torta che mostra la distribuzione dei prodotti nei 3 cluster.

5.2 Apprendimento Supervisionato

5.2.1 Train-Test Split e Bilanciamento

La tabella con il `cluster` di appartenenza diventa il *ground truth* per i modelli di classificazione. Si effettua un *train_ test_ split* (es. 80%-20%) e, in presenza di sbilanciamento, si può usare *SMOTE* per generare esempi sintetici di cluster minoritari.

5.2.2 Modelli Considerati

Nel progetto si addestrano e valutano tre modelli principali:

- **Decision Tree**
- **Random Forest** (potenzialmente ottimizzata con *Optuna*)
- **Logistic Regression**

5.2.3 Valutazione delle Performance

Le metriche chiave includono:

- **Accuracy**
- **F1-score** (macro/weighted)
- **Precision** e **Recall**

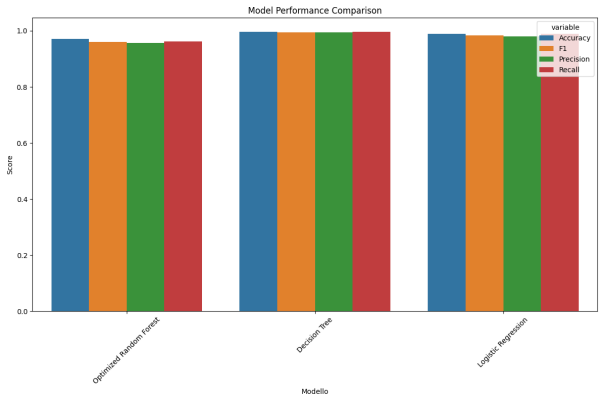

Figura 4: Confronto delle performance (Accuracy, F1, Precision, Recall) tra i principali modelli.

5.2.4 Feature Importance e Analisi SHAP

Per interpretare il comportamento del modello, si ricorre alla:

- **Feature importance** (in particolare, per *Random Forest*);
- **SHAP** (*SHapley Additive exPlanations*), che fornisce un contributo di ogni feature sulla predizione.

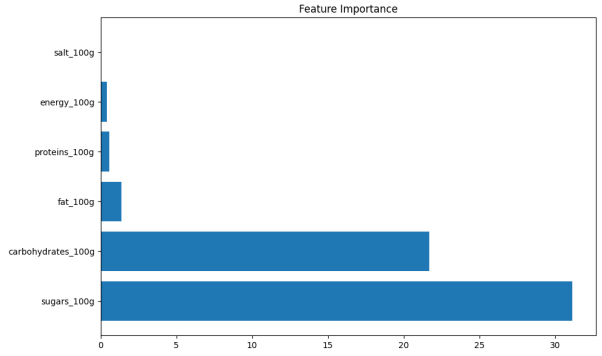

Figura 5: Esempio di *feature importance* per un modello Random Forest.

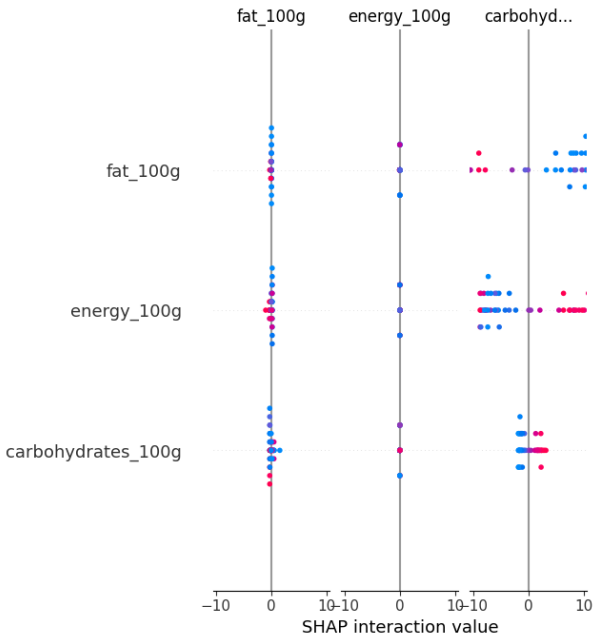

Figura 6: Esempio di *SHAP summary plot* per interpretare i contributi delle variabili.

5.2.5 Learning Curves

Le **learning curve** mostrano come varia la performance all'aumentare dei campioni di training, indicando *bias* o *variance* dei modelli.

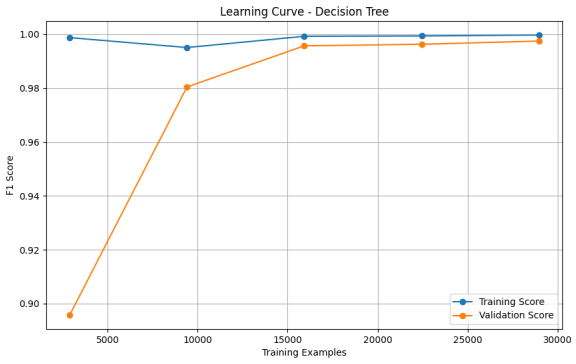

Figura 7: Curva di apprendimento per Decision Tree.

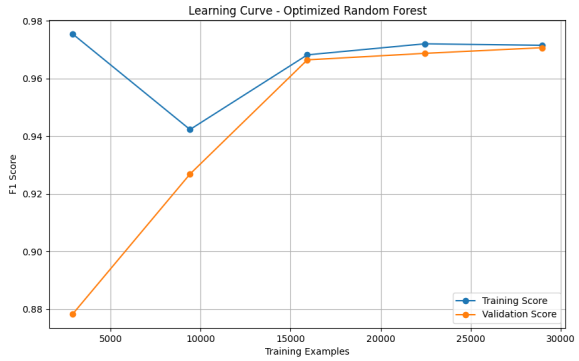

Figura 8: Curva di apprendimento per Random Forest Ottimizzato (con Optuna).

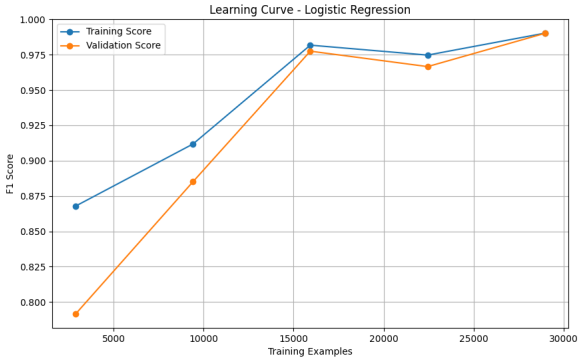

Figura 9: Curva di apprendimento per Logistic Regression.

6 Model Tuning and Enhancement (Ricerca Iperparametri e Reti Bayesiane)

6.1 Ottimizzazione dei Modelli

Per migliorare i risultati, si utilizza **Optuna** (o *GridSearch*) sull'insieme di iperparametri, ad esempio per la **Random Forest**:

- `n_estimators`
- `max_depth`
- `min_samples_split`
- `bootstrap`

L'obiettivo è massimizzare la media del punteggio *F1* o *Accuracy* in *cross-validation*.

6.2 Reti Bayesiane

6.2.1 Rete Bayesiana Continua

Le *Reti Bayesiane* modellano relazioni probabilistiche tra le variabili. Nella forma continua, i valori delle feature non vengono discretizzati. Si utilizza:

- **Hill Climb Search**
- **BIC** (Bayesian Information Criterion)

per apprendere la struttura del grafo (DAG).

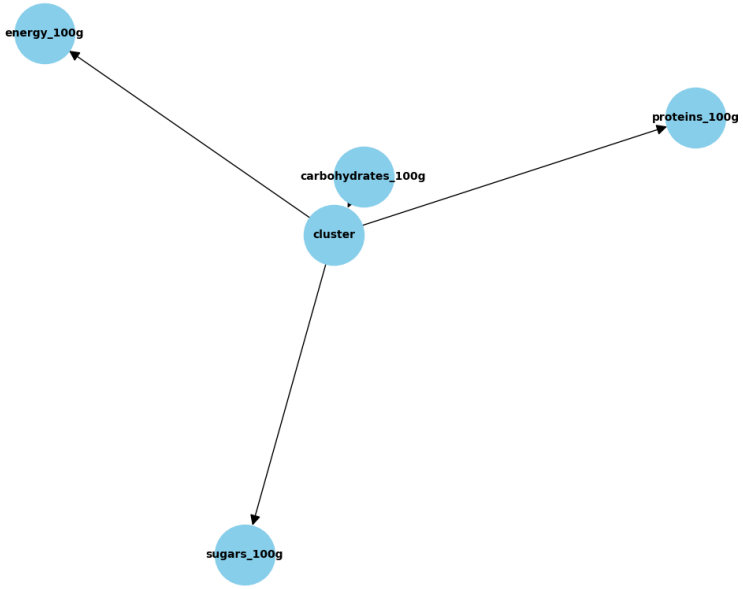

Figura 10: Esempio di rete Bayesiana continua.

6.2.2 Rete Bayesiana Discreta

In alternativa, `KBinsDiscretizer` suddivide le feature in un numero di *bin* (es. 5), prima di stimare la struttura della rete con gli stessi metodi (*Hill Climb*, *BIC*).

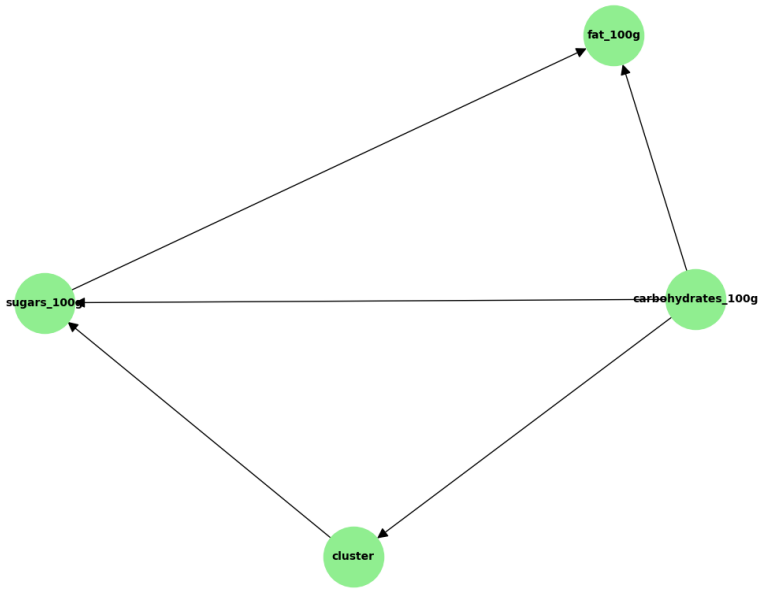

Figura 11: Esempio di rete Bayesiana discretizzata (5 bin).

6.2.3 Inferenza Bayesiana

Le reti Bayesiane così apprese permettono di effettuare:

- **Inferenza** in presenza di dati mancanti;
- Calcolo di *probabilità condizionate* e analisi "what-if";
- Supporto decisionale basato sulle evidenze osservate (es. sugar = "alto", salt = "basso", ecc.).

7 Finalize Model (Salvataggio e Base di Conoscenza Prolog)

7.1 Generazione Automatica di `knowledge_base.pl`

Lo script `generate_prolog_knowledge_base.py` legge dal file `clustered_dataset.csv` e produce fatti Prolog, ad esempio:

```
product(energy, fat, carbs, sugar, protein, salt).
cluster_assignment(energy, fat, carbs, sugar, protein, salt, clusterID).
```

Consentendo di ricostruire o inferire il cluster a partire dalle feature nutrizionali.

7.2 Interfaccia Python–Prolog

Utilizzando `pyswip`, lo script `prolog_interface.py` carica la knowledge base (*consult/1*) e permette query come:

```
?- product_cluster(E, F, C, S, P, Sa, Cluster).
```

restituendo le associazioni `(E,F,C,S,P,Sa) -> Cluster` contenute nella base di conoscenza.

8 Query Esempio in SWI-Prolog

Dopo aver lanciato il comando:

```
?- ['/home/gianuca/Scrivania/ProgettiPersonali/SmartFoodSelector/prolog/knowledge_base
true.
```

è possibile eseguire la query:

```
?- product_cluster(Energy, Fat, Carbohydrates, Sugars, Proteins, Salt, Cluster).
```

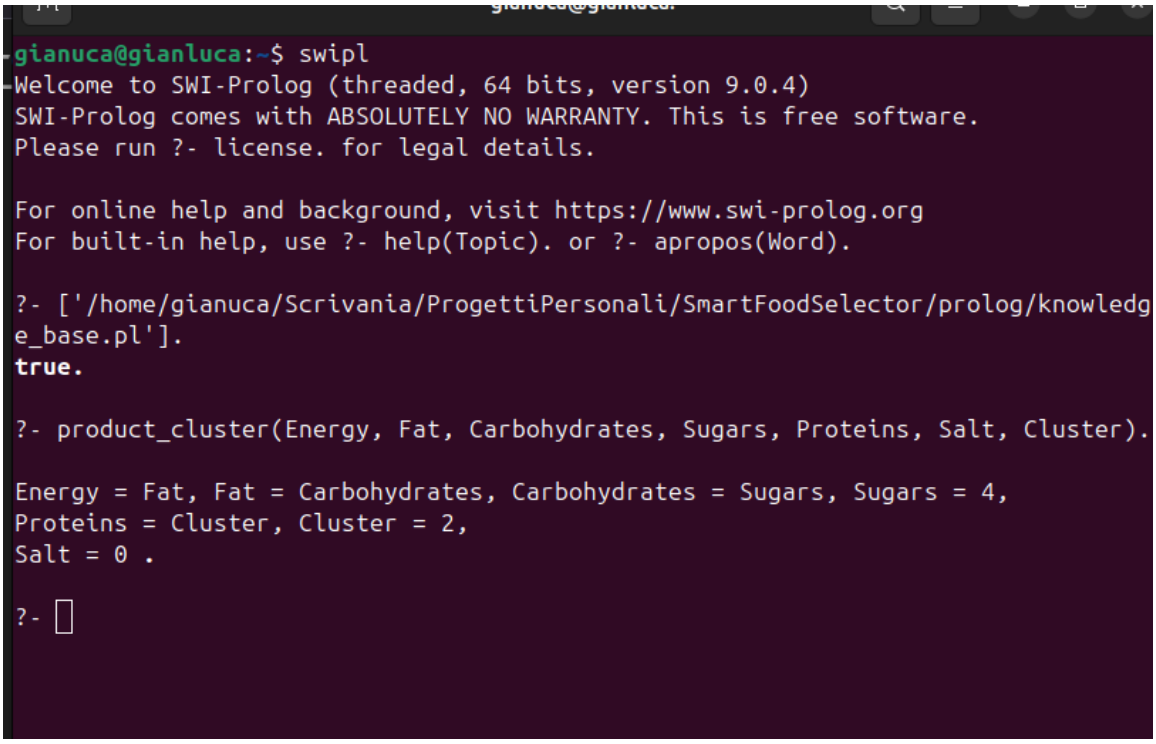

Figura 12: Esempio di interrogazione Prolog e risultato in console.

Come si nota, Prolog risponde indicando l'unificazione tra le variabili (`Energy, Fat, ...`) e i valori corrispondenti trovati nella knowledge base, ad esempio:

```
Energy = Fat, Fat = Carbohydrates, Carbohydrates = Sugars, Sugars = 4,
Proteins = Cluster, Cluster = 2,
Salt = 0 .
```

Il che significa che:

$$Energy = Fat = Carbohydrates = Sugars = 4, \quad Proteins = 2, \quad Salt = 0, \quad Cluster = 2.$$

9 Esempio di Esecuzione e Flusso Principale (`main.py`)

9.1 Fasi Principali

Lo script `main.py` esegue i passi in sequenza:

1. **Preprocessing**: carica e pulisce il dataset (`DataPreprocessor`).
2. **Clustering**: esegue *k-Means* e salva `cluster_assignment`.
3. **Visualizzazione cluster**: genera grafici come l'*Elbow Plot* e la *pie chart*.
4. **Training e valutazione dei modelli**:
   - Suddivisione train/test
   - *SMOTE* per il bilanciamento
   - Addestramento Decision Tree, Random Forest, Logistic Regression

- Metriche e *feature importance*

5. **Reti Bayesiane**: costruite in forma continua e discreta, con i relativi plot.
6. **Generazione base di conoscenza Prolog**.

9.2 Esempio di Console Output

Di seguito si riporta un estratto di esecuzione da riga di comando (ad esempio su ambiente Linux):

```
gianuca@gianluca:~/Scrivania/ProgettiPersonali/SmartFoodSelector$ source venv/bin/acti
(venv) gianuca@gianluca:~/Scrivania/ProgettiPersonali/SmartFoodSelector$ python main.p
/home/gianuca/.../main.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release ...

Inizio del programma...
1) Preprocessing dei dati...
Preprocessing completato, dati salvati in: data/processed/Dataset.csv
Preprocessing completato in 13.26 secondi.

2) Clustering dei dati...
Assegnazione cluster completata, salvato in: data/results/clustered_dataset.csv
Clustering completato in 1.74 secondi.

Generazione del grafico a torta per il clustering...
3) Training e valutazione dei modelli...
Inizio ottimizzazione dei parametri del modello Random Forest...
[ ... Output con i trial di Optuna ... ]
Ottimizzazione completata!
I migliori parametri trovati sono:
 - n_estimators: 141
 - max_depth: 9
 - min_samples_split: 0.40321021451655503
 - bootstrap: True
Cross-validation per il modello Optimized Random Forest...
F1 scores: ...
Mean F1: ...
Std Dev: ...

4) Creazione della rete bayesiana...
Creazione della rete bayesiana con valori continui...
[ ... log di Hill Climb Search e BIC ... ]
Grafico salvato in: data/results/plots/bayesian_network_continuous.png
Creazione della rete bayesiana con valori discreti...
[ ... log di Hill Climb Search e BIC ... ]
Grafico salvato in: data/results/plots/bayesian_network_discrete.png
Programma completato in 109.73 secondi.
```

10 Risultati Principali e Osservazioni

10.1 Prestazioni dei Modelli Supervisionati

I classificatori (Decision Tree, Random Forest, Logistic Regression) hanno ottenuto **buone performance** in termini di *accuracy*, *F1-score*, *precision* e *recall*. Particolarmente efficace è risultata la **Random Forest**, soprattutto dopo il tuning con *Optuna*.

10.2 Vantaggi delle Reti Bayesiane

- Gestione dell'incertezza (mancanza parziale di dati);
- Modellazione delle dipendenze probabilistiche in un **grafo aciclico**;
- Analisi di scenari "what if".

10.3 Integrazione con Prolog

La **knowledge base** generata (file `knowledge_base.pl`) consente interrogazioni logiche e ragionamenti su regole e pattern, arricchendo le capacità puramente *statistiche* con una componente *simbolica*.

11 Conclusioni

Il progetto **SmartFoodSelector** dimostra come combinare tecniche di *Machine Learning* (clustering, classificazione, reti bayesiane) con un sistema di *logica dichiarativa* (Prolog). Le principali funzionalità possono essere riassunte in:

- Pipeline completa dalla pulizia dati, al *k-Means*, a modelli *supervisionati* con *hyperparameter tuning*;
- Creazione di **Reti Bayesiane** (continue/discrete) per inferenza probabilistica;
- Generazione di una **base di conoscenza Prolog** per query e ragionamenti logici.

L'architettura modulare rende il sistema facilmente manutenibile ed estensibile ad altri scenari di analisi nutrizionale.