Documentazione Completa del Progetto SmartFoodSelector

Gianluca Lascaro matricola 75867

26 gennaio 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 Introduzione

SmartFoodSelector è un sistema modulare progettato per:

- **Effettuare preprocessamento** dei dati nutrizionali dei prodotti (es. energy, fat, sugar, protein);
- **Suddividere i prodotti** in gruppi omogenei (*cluster*) in base a caratteristiche nutrizionali simili, tramite *k-Means*;
- **Addestrare modelli** di classificazione supervisionata per assegnare rapidamente un nuovo prodotto al cluster più adatto;
- **Strutturare relazioni di dipendenza probabilistica** (Reti Bayesiane) per inferenze anche in presenza di valori mancanti;
- **Fornire un'interfaccia Prolog** in grado di rappresentare una base di conoscenza e rispondere a query logiche.

L'obiettivo centrale è fornire uno strumento integrato di *supporto decisionale* nel dominio alimentare, combinando aspetti di data science con tecniche di intelligenza artificiale simbolica.

2 Architettura Generale del Progetto

Il progetto è organizzato con una struttura a *pacchetti* e *moduli* Python, affiancati da una base di conoscenza Prolog generata in modo semi-automatico:

- `src/dataset_preprocessing.py`: modulo per il **preprocessing** (pulizia, selezione feature, normalizzazione).
- `src/unsupervised_clustering.py`: modulo che implementa il **k-Means** e genera eventuali grafici per mostrare la "elbow curve" e la distribuzione dei cluster.
- `src/supervised_trainer.py` (o nome analogo): modulo dedicato al **training** di modelli di classificazione come *Decision Tree*, *Random Forest* e *Logistic Regression*, con valutazioni di *accuracy*, *F1* e altri indicatori.
- `src/bayes_net.py`: modulo per la **creazione di Reti Bayesiane** (in forma sia continua sia discretizzata) e per la visualizzazione del grafo delle dipendenze apprese.
- `src/prolog_interface.py`: fornisce metodi per **caricare la knowledge base** in Prolog, eseguire query e gestire i risultati.
- `src/generate_prolog_knowledge_base.py`: script che genera automaticamente la base di conoscenza `knowledge_base.pl`, contenente fatti e regole relative ai prodotti e ai cluster.
- `main.py`: **script principale** che coordina l'esecuzione delle varie fasi (preprocessing, clustering, training modelli, costruzione rete bayesiana, generazione base di conoscenza).

La scelta di suddividere il progetto in più moduli garantisce *manutenibilità*, *riusabilità* e *scalabilità* dell'applicazione.

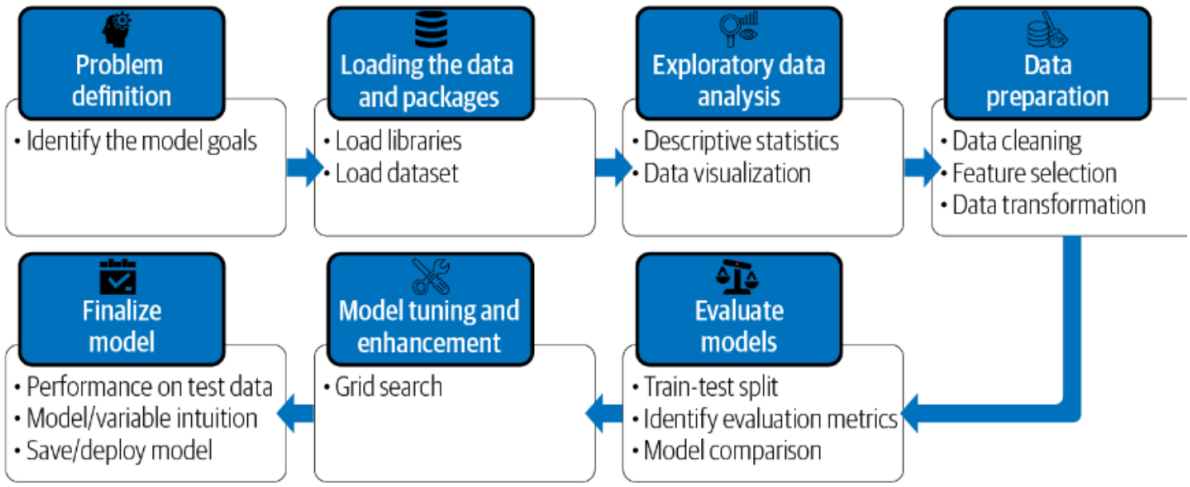

Figura 1: Model development steps

3 Preprocessing dei Dati

3.1 Caricamento e Pulizia

In questa fase i dati (ad es. dal file `en.openfoodfacts.org.products.tsv`) vengono caricati in un `DataFrame`. Si procede con l'eventuale **rimozione di colonne** non utili e con la pulizia dei **valori mancanti** o chiaramente errati. Nel progetto è stata effettuata una selezione delle variabili essenziali:

energy_100g, fat_100g, carbohydrates_100g, sugars_100g, proteins_100g, salt_100g.

3.2 Normalizzazione con MinMaxScaler

Per uniformare l'ordine di grandezza delle feature, si applica il *MinMaxScaler* di `sklearn`, che trasforma i valori in un intervallo [0,1]. Questo step rende il dataset più *bilanciato* rispetto a variabili con scale molto diverse (es. `salt_100g` vs. `energy_100g`).

3.3 Salvataggio in Formato CSV

Una volta pre-processato, il dataset viene esportato in un file CSV (es. `newDataset.csv`), pronto per le fasi di *clustering* e *training*.

4 Clustering Non Supervisionato

4.1 Algoritmo k-Means

Si utilizza l'algoritmo *k-Means* per individuare **gruppi di prodotti** con caratteristiche nutrizionali simili. Il metodo consiste nel:

1. Scegliere un valore k (numero di cluster).
2. Inizializzare casualmente k centroidi.
3. Assegnare ciascun campione al centroide più vicino.
4. Aggiornare i centroidi come media dei punti assegnati.
5. Ripetere i passi finché la soluzione converge.

4.2 Scelta del Numero di Cluster: Elbow Plot

Per decidere k, si applica il **metodo del gomito** (*Elbow Method*), che valuta l'*inerzia* (valore cumulativo dell'errore quadratico) in funzione di k. Nel progetto si utilizza la libreria `kneed` per determinare il "ginocchio" (elbow).

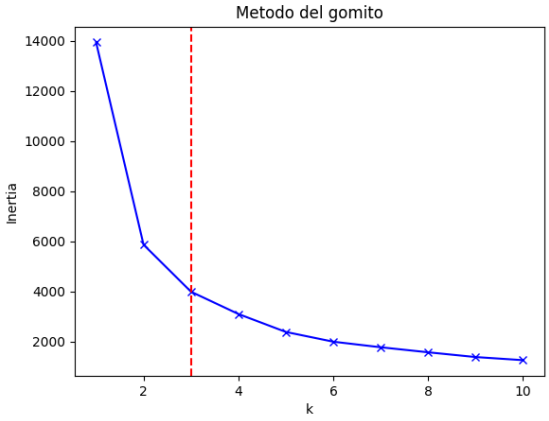

Figura 2: Esempio di *Elbow Plot* che suggerisce la scelta di $k = 3$.

4.3 Assegnazione e Distribuzione dei Cluster

Confermato il valore $k = 3$, si esegue *k-Means* e si aggiunge una colonna `cluster` al dataset, indicando la *classe* di appartenenza di ogni prodotto (0, 1, 2).

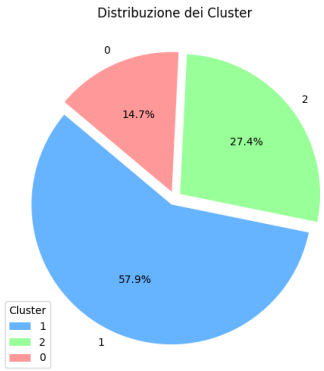

Figura 3: Grafico a torta che illustra la distribuzione dei prodotti nei tre cluster.

Il file di output comunemente viene salvato come `clustered_dataset.csv`.

5 Apprendimento Supervisionato

5.1 Obiettivi

Dopo aver etichettato i prodotti con un cluster (fase di clustering), si costruiscono modelli supervisionati in grado di *prevedere* il cluster di un nuovo prodotto a partire dalle sue caratteristiche nutrizionali, senza dover rieseguire il clustering su tutto il dataset.

5.2 Modelli Considerati

Nel progetto **SmartFoodSelector**, i classificatori adottati includono:

- **Decision Tree Classifier**
- **Random Forest Classifier** (ensemble di alberi, eventualmente ottimizzati con *Optuna*)
- **Logistic Regression**

5.3 Train-Test Split e Tecniche di Bilanciamento

Prima del training, il dataset viene suddiviso (*train_ test_ split*) in dati di addestramento e di test. Per gestire sbilanciamenti, si impiega *SMOTE* (Synthetic Minority Over-sampling Technique).

5.4 Valutazione delle Performance

Le **metriche** calcolate includono:

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

Si effettua inoltre una *valutazione comparativa* tra i vari modelli per scegliere il più performante.

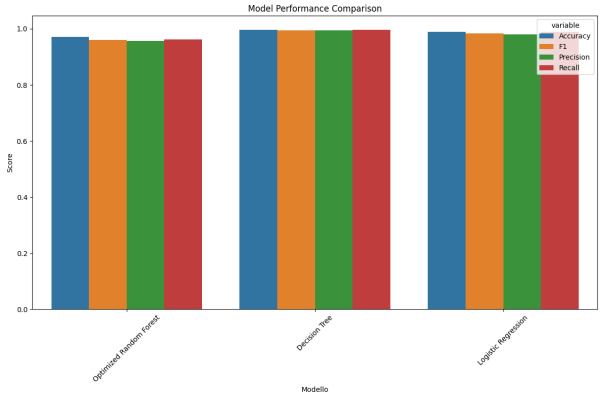

Figura 4: Confronto delle metriche di performance dei principali modelli di classificazione.

5.5 Importanza delle Feature e Analisi SHAP

Per interpretare i risultati:

- Viene calcolata la **feature importance** (soprattutto in modelli ad alberi).
- Viene utilizzata **SHAP (SHapley Additive exPlanations)** per spiegare l'impatto di ogni variabile sulle previsioni del modello.

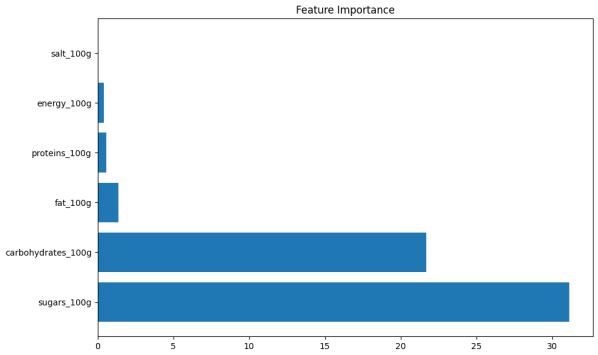

Figura 5: Importanza delle feature per un modello di tipo Random Forest.

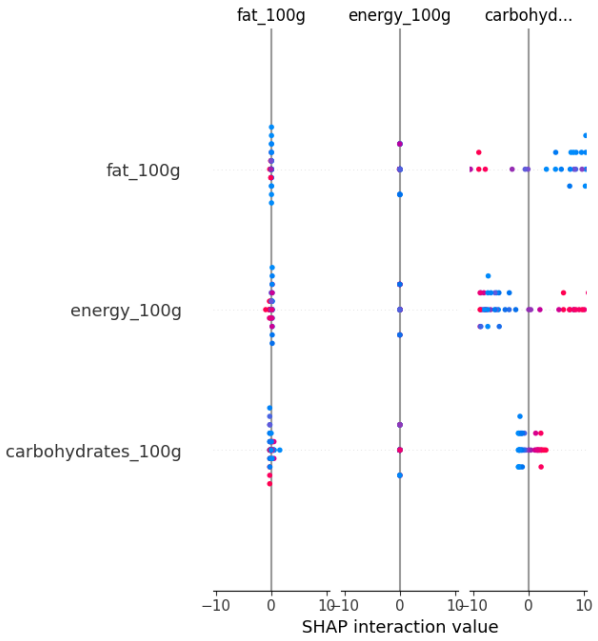

Figura 6: Esempio di *SHAP Summary Plot* per interpretare i contributi delle feature.

5.6 Curve di Apprendimento (Learning Curves)

Le **Learning Curves** mostrano come varia la performance dei modelli al crescere della dimensione del training set.

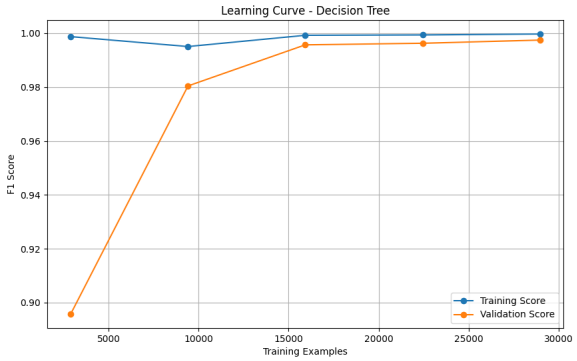

Figura 7: Curva di apprendimento per Decision Tree.

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

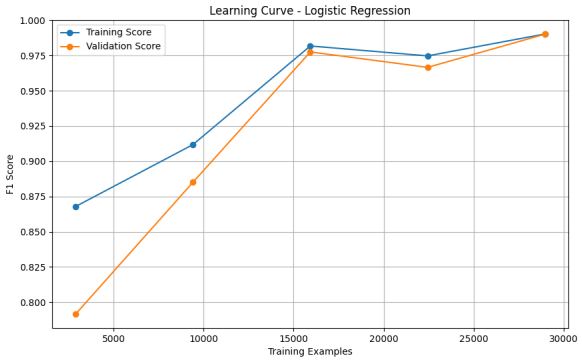

Figura 9: Curva di apprendimento per Logistic Regression.

6 Reti Bayesiane

6.1 Rete Bayesiana Continua

Le *Reti Bayesiane* consentono di modellare **relazioni probabilistiche** tra variabili. In una **rete continua**, i valori delle variabili sono trattati come reali (senza discretizzazione). Nel progetto, la struttura è appresa con:

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

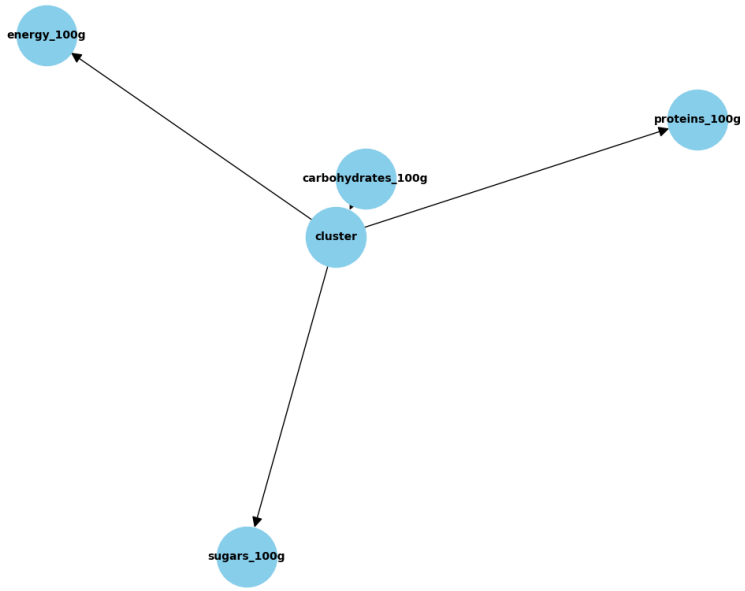

Figura 10: Rete Bayesiana appresa con valori *continui*.

6.2 Rete Bayesiana Discreta

In alternativa, si può **discretizzare** le variabili (ad es. con `KBinsDiscretizer`, 5 bin), prima di eseguire l'apprendimento della struttura bayesiana.

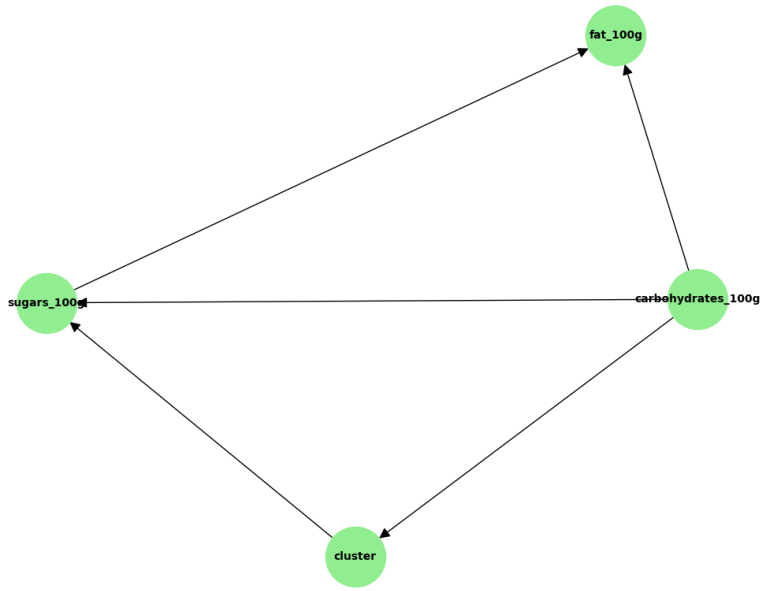

Figura 11: Rete Bayesiana con valori *discretizzati* in 5 bin.

6.3 Applicazioni e Inferenza

Le reti bayesiane così costruite permettono di:

- Fare inferenza probabilistica in presenza di dati parziali o mancanti;
- Calcolare distribuzioni di probabilità congiunte e condizionate;
- Supportare decisioni in base a **evidenze** su determinate variabili (es. sugar, salt).

7 Generazione e Interrogazione della Knowledge Base (Prolog)

7.1 Generazione Automatica del File `knowledge_base.pl`

Lo script apposito (`generate_prolog_knowledge_base.py`) legge il dataset clusterizzato (`clustered_dataset.csv`) e:

- Scrive fatti `product(...)` per ciascuna riga (valori nutrizionali).
- Scrive fatti `clustered_product(...)` che associano il prodotto a un particolare *cluster*.
- Definisce una regola `product_info(...)` che ricostruisce il cluster a partire dai valori nutrizionali.

7.2 Interfaccia Python–Prolog

Grazie alla libreria `pyswip`, si *consulta* la knowledge base Prolog (*consult/1*) e si eseguono query. Esempio di query di test:

```
?- member(X, [a, b, c]).
```

che stampa i risultati trovati ($X = a, X = b, X = c$) verificando la corretta configurazione dell'ambiente Prolog.

7.3 Esempio di Query Applicativa

```
?- product_info(E, F, C, Su, P, Sa, Cluster).
```

L'interprete Prolog restituirà tutte le combinazioni di $\{E, F, C, Su, P, Sa\}$ (valori nutrizionali) con il corrispondente *cluster*.

8 Coordinamento del Flusso (`main.py`)

Lo script principale `main.py` esegue in sequenza le fasi:

1. **Preprocessing** (caricamento file TSV, pulizia, normalizzazione, salvataggio CSV).
2. **Clustering** (k-Means) e scelta del numero di cluster (*elbow_plot*).

3. **Visualizzazione** della distribuzione dei cluster (pie chart).
4. **Training & Evaluate** modelli di classificazione:
    - Suddivisione train/test
    - Bilanciamento con SMOTE
    - Addestramento & cross-validation (Decision Tree, Random Forest, Logistic Regression)
    - Salvataggio metriche e plot di performance
5. **Costruzione Rete Bayesiana** in forma continua e discreta, inclusa la generazione dei plot.
6. **Generazione Knowledge Base Prolog** (opzionale).

Alla fine, si ottengono:

- **Dataset preprocessato** (`.csv`);
- **Dataset clusterizzato** (`.csv`);
- **Modelli supervisionati** addestrati e valutati (con grafici e metriche);
- **Reti bayesiane** (grafici e parametri stimati);
- **Knowledge base Prolog** per query logiche su prodotti e cluster.

9 Risultati Principali e Osservazioni

9.1 Prestazioni dei Modelli Supervisionati

I modelli (Decision Tree, Random Forest, Logistic Regression) hanno mostrato **ottime performance** in termini di *accuracy*, *F1* e *precision/recall*. La combinazione di *clustering* preliminare e training supervisionato facilita la categorizzazione di nuovi prodotti con elevata affidabilità.

9.2 Vantaggi dell'Approccio Bayesiano

L'uso di **Reti Bayesiane** permette di:

- Gestire in modo *naturale* l'incertezza e i dati mancanti;
- Rappresentare relazioni di dipendenza complesse in forma di *grafo aciclico*;
- Effettuare **inferenza probabilistica** e *what-if analysis*.

9.3 Integrazione con Prolog

La **base di conoscenza Prolog** rende possibile compiere *query simboliche* (es. regole, ricerche di pattern) che un sistema puramente statistico faticherebbe a gestire senza un'adeguata formalizzazione logica.

10 Conclusioni

Il progetto **SmartFoodSelector** dimostra come *tecniche di Machine Learning*, *Inferenza Bayesiana* e *logica Prolog* possano lavorare in sinergia per offrire un sistema robusto e modulare di analisi nutrizionale e di supporto decisionale.