Burned Calories Estimation Model

A.A. 2024/2025

Di Chio Maria Teresa

m.dichio9@studenti.uniba.it

Link GitHub:
`https://github.com/iamariateresa7/Burned-Calories-Estimation-Model`

Prof. Nicola Fanizzi

Contents

1 Introduzione

L'analisi e la previsione del consumo calorico durante l'attività fisica rappresentano un aspetto cruciale per il monitoraggio della salute e delle prestazioni atletiche.
Questo progetto si propone di sviluppare un modello predittivo in grado di stimare le calorie bruciate da un utente durante una sessione di allenamento, basandosi su caratteristiche personali e parametri dell'esercizio svolto.

Il progetto è stato realizzato in `Python`, in particolare la `versione 3.12.2`.
Per l'avvio, basta eseguire il file `predict_models.py` all'interno del progetto.

1.1 Argomenti di Interesse

- Apprendimento supervisionato (Regressione Lineare, Random Forest, Gradient Boosting con Grid Search CV)
- Apprendimento probabilistico (Regressione Bayesiana, SGDClassifier con Grid Search CV)
- Knowledge Base (KB) per supportare il processo decisionale e l'inferenza basata su regole

1.2 Librerie Utilizzate

Di seguito sono elencate le principali librerie utilizzate nel progetto, con una breve descrizione delle loro funzionalità.

- **pandas**: libreria per la manipolazione e l'analisi dei dati in modo efficiente;
- **joblib**: utilizzata per il salvataggio e il caricamento di oggetti Python;
- **scikit-learn**: libreria di machine learning che fornisce strumenti per la modellazione predittiva, tra cui algoritmi di regressione, classificazione, clustering e riduzione della dimensionalità. Include anche strumenti per la valutazione delle prestazioni dei modelli e la trasformazione dei dati;
- **numpy**: libreria per il calcolo numerico ad alte prestazioni, utilizzata per operazioni su array e calcolo di metriche;
- **json**: libreria standard di Python per la gestione di file JSON, utile per salvare configurazioni e risultati in formato strutturato;
- **matplotlib.pyplot**: una libreria per la creazione di grafici e visualizzazioni statiche in Python;
- **os**: modulo che fornisce una modo per interagire con il sistema operativo, utile per la gestione di file e cartelle;

- **sys**: fornisce funzioni e variabili per interagire con l'interprete, come la gestione degli argomenti da riga di comando, il controllo del flusso di output e l'accesso al sistema operativo;
- **pyswip (Prolog)**: libreria che permette l'integrazione di Prolog in Python, utile per la gestione di una Knowledge Base (KB) e per eseguire inferenze logiche basate su regole.

2 Creazione del dataset

Il dataset utilizzato in questo progetto è stato scaricato dalla piattaforma `https://www.kaggle.com/`, si trova all'interno della cartella `dataset` ed il suo nome è `gym_members_exercise_tracking.csv`.
Questo dataset contiene informazioni su sessioni di allenamento di diversi utenti, registrando vari parametri fisiologici e dettagli dell'attività fisica svolta.

Le colonne principali sono descritte di seguito:

- **Age**: età dell'utente in anni;
- **Gender**: genere dell'utente (*Male* per maschio, *Female* per femmina);
- **Weight (kg)**: peso dell'utente espresso in chilogrammi;
- **Height (m)**: altezza dell'utente espressa in metri;
- **Max_BPM**: frequenza cardiaca massima registrata durante l'allenamento (battiti per minuto);
- **Avg_BPM**: frequenza cardiaca media durante l'allenamento (battiti per minuto);
- **Resting_BPM**: rfequenza cardiaca a riposo (battiti per minuto);
- **Session_Duration (hours)**: durata della sessione di allenamento, espressa in ore;
- **Calories_Burned**: calorie bruciate durante la sessione di allenamento;
- **Workout_Type**: tipo di allenamento svolto (es. *Yoga*, *HIIT*, ecc.);
- **Fat_Percentage**: percentuale di massa grassa dell'utente.
- **Water_Intake (liters)**: Quantità di acqua assunta durante la sessione, espressa in litri;
- **Workout_Frequency (days/week)**: frequenza degli allenamenti settimanali (numero di giorni a settimana);
- **Experience_Level**: livello di esperienza dell'utente nell'attività fisica (*1* per principiante, *2* per intermedio, *3* per avanzato);

- **BMI**: indice di massa corporea (*Body Mass Index*), calcolato come $\frac{\text{peso (kg)}}{\text{altezza (m)}^2}$.

Questo dataset permette di analizzare l'impatto di vari fattori fisiologici e delle abitudini di allenamento sul consumo calorico e sulle prestazioni fisiche degli utenti.

2.1 Pulizia e Preprocessing dei dati

La pulizia dei dati è una fase fondamentale del progetto, in quanto garantisce che il dataset sia coerente e privo di anomalie prima di essere utilizzato per l'addestramento del modello.

La pulizia e il prepocessing dei dati si trovano all'interno della cartella `dataset`, nel file `dataset_utils.py`.

Di seguito vengono descritti i principali passaggi eseguiti per preparare i dati.

Rinomina delle Colonne

Nel dataset originale, alcune colonne contenevano spazi nei loro nomi.
Per facilitare l'accesso ai dati e migliorare la compatibilità con i framework di machine learning, ho sostituito gli spazi con il carattere _.
Questo assicura che ogni colonna abbia un nome uniforme e facilmente utilizzabile nel codice.

```
df.columns = [col.replace(' ', '_') for col in df.columns]
```

Conversione della Variabile "Workout_Type" in Categoria

La colonna `Workout_Type` contiene valori testuali che rappresentano il tipo di allenamento svolto dall'utente.
Per ottimizzare il modello, è stato necessario convertirla in una variabile categorica. Questa operazione assegna a ciascun tipo di allenamento un codice numerico, ma per garantire la possibilità di riconvertire i dati, ho creato un mapping tra i codici e le categorie originali.
Il mapping viene salvato su disco `(workout_mapping.pkl)` all'interno della cartella `modelli` dei due tipi di addestramenti e potrà essere utilizzato in fase di predizione per riconvertire i codici numerici nei nomi originali.

```
df['Workout_Type'] = df['Workout_Type'].astype('category')
workout_mapping = dict(enumerate(df['Workout_Type'].cat.categories))
joblib.dump(workout_mapping, 'workout_mapping.pkl')
df['Workout_Type'] = df['Workout_Type'].cat.codes
```

Conversione delle Variabili Categoriali

Un'altra colonna contenente valori testuali è `Gender`, che indica il genere dell'utente (Male o Female).
Poiché i modelli di machine learning lavorano meglio con dati numerici, ho convertito questa variabile in numeri.
Ora Gender = 0 rappresenta "Male" e Gender = 1 rappresenta "Female".

```
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
```

Standardizzazione dei dati

La standardizzazione è una tecnica di *preprocessing* che viene utilizzata per portare tutte le variabili numeriche a una scala comune.
In pratica, consiste nel trasformare ogni feature del dataset affinché abbia media zero e deviazione standard unitaria. Questo significa che ogni variabile avrà una distribuzione normale (se non lo è già) con media 0 e deviazione standard 1.

```
def standardize_features(X_train, X_test, scaler_path="modelli/scaler.pkl"):
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    joblib.dump(scaler, scaler_path)
    return pd.DataFrame(X_train_scaled, columns=X_train.columns), pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

La standardizzazione è importante perché può capitare che le feature abbiano scale diverse (ad esempio, età in anni e peso in chilogrammi) e di conseguenza i modelli di machine learning possono dare più importanza alle feature con valori numerici più grandi, come il peso.
La standardizzazione rende tutte le feature equivalenti in termini di scala, evitando che alcune influenzino il modello più di altre.

3 Apprendimento Supervisionato

L'obiettivo principale è far sì che l'algoritmo impari una relazione tra gli input e gli output in modo che, una volta addestrato, possa fare previsioni accurate su nuovi dati mai visti prima.

Ci sono diverse fasi da affrontare:

- Scelta degli iper-parametri;
- Fase di addestramento;
- Fase di test;
- Valutazione delle prestazioni.

Suddivisione del Dataset e Standardizzazione

Dopo la preparazione iniziale, vengono selezionate le **feature** (variabili indipendenti) e la **variabile target** da predire, ovvero le calorie bruciate.

Le feature utilizzate sono:

- **Gender**: genere dell'utente (*Male* per maschio, *Female* per femmina);
- **Workout_Type**: tipo di allenamento svolto (es. *Yoga*, *HIIT*, ecc.);
- **Session_Duration_(hours)**: durata della sessione di allenamento, espressa in ore;
- **Weight_(kg)**: peso dell'utente espresso in chilogrammi;
- **Age**: età dell'utente in anni;
- **Avg_BPM**: frequenza cardiaca media durante l'allenamento (battiti per minuto).

Ho scelto queste feature perché influiscono maggiormente nella predizione delle calorie bruciate.

La variabile target è **Calories_Burned** ovvero le calorie bruciate durante la sessione di allenamento.

Il dataset viene quindi suddiviso in due parti:

1. un **training set** (0.8) per addestrare i modelli;
2. un **test set** (0.2) per valutarne le prestazioni.

3.1 Addestramento dei Modelli

Vengono addestrati tre modelli di apprendimento supervisionato:

1. un modello di **regressione lineare**
2. un modello basato su **Random Forest**
3. un modello di **Gradient Boosting**

3.2 Regressione Lineare

La regressione lineare viene utilizzata come primo approccio per stabilire una relazione tra le variabili indipendenti e la variabile target.
Questo metodo fornisce una prima stima della capacità predittiva del dataset.

La regressione lineare è un modello statistico utilizzato per predire una variabile dipendente in base a una o più variabili indipendenti.
In questo progetto, è stata impiegata per stimare le calorie bruciate durante gli allenamenti in palestra in funzione di caratteristiche dell'utente e del tipo di esercizio.

3.2.1 Scelte progettuali

Questo modello è stato addestrato utilizzando la libreria `scikit-learn` senza l'uso di una Grid Search per l'ottimizzazione degli iperparametri.
Questo perché la regressione lineare ha pochi parametri regolabili e generalmente non richiede una ricerca estensiva degli iperparametri come avviene per modelli più complessi (ad esempio Random Forest e Gradient Boosting).

3.2.2 Valutazione del modello

Per valutare le prestazioni del modello, sono state calcolate **due metriche** principali:

- **Mean Absolute Error (MAE)**: misura la differenza media assoluta tra i valori predetti e quelli reali;
- **Root Mean Squared Error (RMSE)**: valuta la differenza quadratica media tra le predizioni e i valori reali, penalizzando maggiormente gli errori più grandi.

Per verificare la presenza di **overfitting**, le metriche sono state calcolate sia sul set di test che sul set di training.

3.2.3 Overfitting

L'overfitting si verifica quando un modello di machine learning si adatta troppo bene ai dati di training, al punto da perdere la capacità di generalizzare su dati nuovi (test set).
In pratica, il modello "memorizza" i dettagli e il rumore del training set invece di apprendere le relazioni generali tra le variabili, risultando in prestazioni eccellenti sul training set ma scarse su dati mai visti prima.

Per valutare la presenza di overfitting, si possono utilizzare differenti modi, come calcolare la *deviazione standard* o la *varianza* e analizzare i risultati.

In questo progetto, ho deciso di confrontare le metriche di errore (come MAE e RMSE) calcolate sul training set e sul test set.
Si hanno diversi casi:

- Se le metriche sono simili, il modello ha buone capacità di generalizzazione;
- Se l'errore sul training è molto più basso rispetto al test, il modello potrebbe essere overfittato. Questo significa che ha imparato i dettagli del training set, ma non riesce a fare buone previsioni su nuovi dati;
- Se l'errore è alto sia su training che su test, il modello è probabilmente underfittato, ossia troppo semplice per catturare le relazioni nei dati.

3.2.4 Risultati ottenuti

I risultati ottenuti dal modello di regressione lineare sono i seguenti.

Per la valutazione sul Test Set:

- **MAE**: 29.32
- **RMSE**: 39.47

Mentre per la valutazione sul Training Set:

- **MAE**: 29.90
- **RMSE**: 39.49

Dai risultati si nota che il modello ha prestazioni simili su training e test set, suggerendo un basso rischio di overfitting.
Tuttavia, i valori di errore relativamente alti indicano che la regressione lineare potrebbe non essere il modello più adatto per questa tipologia di dati.

La regressione lineare ha fornito una baseline per la previsione delle calorie bruciate, con risultati accettabili ma migliorabili.
Modelli più complessi come Random Forest e Gradient Boosting hanno permesso di ottenere prestazioni significativamente migliori, suggerendo che la relazione tra le variabili del dataset e il target non è strettamente lineare.

3.3 Random Forest

Il secondo modello analizzato è il Random Forest. Esso è un algoritmo di apprendimento supervisionato basato su un insieme di **alberi decisionali**.
Questo metodo utilizza il concetto di *bagging* (Bootstrap Aggregating) per costruire molteplici alberi decisionali, ognuno addestrato su un sottoinsieme casuale del dataset originale.
La previsione finale viene ottenuta combinando i risultati di tutti gli alberi, riducendo così la varianza e migliorando la robustezza del modello rispetto a un singolo albero decisionale.

3.3.1 Iperparametri di Random Forest

Gli **iperparametri** sono valori che vengono impostati prima dell'addestramento di un modello e influenzano le prestazioni e la capacità di generalizzazione dello stesso.
Nel caso di Random Forest, alcuni degli iperparametri più importanti sono:

- **n_estimators**: il numero di alberi nella foresta. Un numero maggiore di alberi può migliorare la stabilità delle predizioni ma aumenta il costo computazionale.
- **max_depth**: la profondità massima degli alberi. Un valore troppo alto può portare a overfitting, mentre un valore troppo basso può risultare in un modello poco espressivo.
- **min_samples_split**: il numero minimo di campioni richiesti per suddividere un nodo. Valori più alti impediscono una crescita eccessiva degli alberi, riducendo il rischio di overfitting.
- **min_samples_leaf**: il numero minimo di campioni che un nodo foglia deve contenere. Impedisce la creazione di foglie con pochi dati, rendendo il modello più generalizzabile.
- **max_features**: il numero massimo di caratteristiche considerate per ciascuna suddivisione. Può essere impostato su valori come *sqrt* (radice quadrata del numero di feature), *log2* o un numero specifico.
- **bootstrap**: se impostato su *True*, consente il riutilizzo di dati nei diversi alberi attraverso il campionamento con sostituzione.

```
'random_forest': {
    'n_estimators': [200, 300],
    'max_depth': [10, 15],
    'min_samples_split': [20, 30],
    'min_samples_leaf': [5, 10],
    'max_features': ['sqrt', 'log2'],
    'bootstrap': [True, False]
```

3.3.2 Ottimizzazione degli Iperparametri con Grid Search e Cross Validation

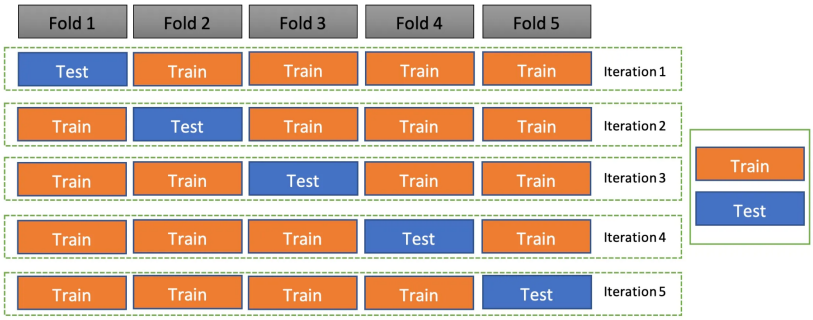
Per trovare la combinazione ottimale degli iperparametri, è stata utilizzata la tecnica della **Grid Search** con **Cross Validation**.

La **Grid Search** consiste nel testare sistematicamente tutte le combinazioni di valori possibili per gli iperparametri specificati, valutando le prestazioni di ciascuna combinazione attraverso la cross validation.

La **Cross Validation** (CV) è un metodo che divide il dataset in più sottoinsiemi (**folds**), utilizzando alcuni di essi per l'addestramento e altri per la validazione, riducendo così il rischio di overfitting e fornendo una stima più affidabile delle prestazioni del modello.

Nel nostro caso, è stata utilizzata una Grid Search con **5-fold cross validation**, il che significa che il dataset è stato suddiviso in 5 parti e il modello è stato addestrato e validato 5 volte, usando ogni volta una parte diversa come set di test.

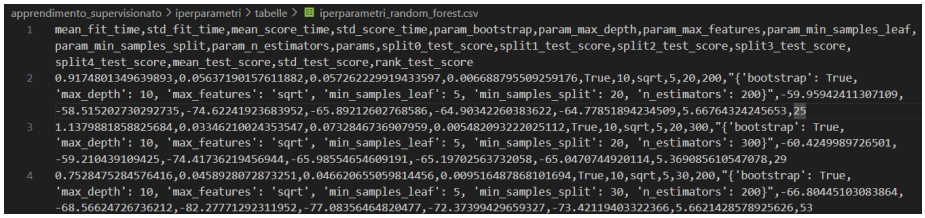
Inizialmente, era stata implementata una Grid Search con **3-fold cross validation** ma era risultata troppo bassa. Ho deciso quindi di aumentare il numero di fold e di utilizzarne 5 per una maggiore affidabilità.
Un valore più alto di cv (ad esempio 5 o 10) garantisce che il modello venga valutato su più combinazioni di set di addestramento e test, migliorando così la stima delle performance.



3.3.3 Creazione della Tabella dei Risultati

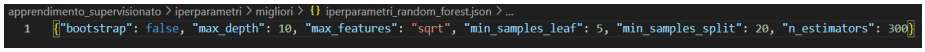
Una volta completata la Grid Search, i risultati ottenuti vengono salvati in una tabella, che mostra le **combinazioni di iperparametri** testate e il punteggio ottenuto per ciascuna.
Questo permette di identificare facilmente la configurazione migliore da utilizzare per il modello finale.

La tabella è stata salvata nella cartella `iperparametri`, nella sottocartella `tabelle` ed il file si chiama `iperparametri_random_forest.csv` che si trova all'interno del progetto.
Il formato CSV è perfetto quando si ha una tabella di dati strutturati in righe e colonne.
Di seguito vediamo una parte della tabelle presente nel file precedente:



I valori di mean_test_score sono negativi (questo è dato dal fatto che stiamo usando *neg_mean_absolute_error* perché GridSearchCV cerca di massimizzare la metrica di scoring, e poiché il MAE è un valore da minimizzare, scikit-learn lo converte in negativo), quindi bisogna sempre considerare il valore assoluto per capire quanto bene il modello sta performando.
Ad esempio, un mean_test_score di -58.16 significa un MAE di 58.16, che è peggiore rispetto ad altri parametri con valori più bassi come -31.4.

Inoltre, i **migliori iperparametri** trovati (combinazione più performante) vengono salvati in un file JSON denominato `iperparametri_random_forest.json` nella cartella `iperparametri`, nella sottocartella `migliori`.
Il formato JSON è ideale per salvare dati strutturati che consistono in coppie chiave-valore.



Questi parametri indicano un modello abbastanza semplice e flessibile.

La scelta degli iperparametri giusti ha un impatto significativo sulle prestazioni del modello e la metodologia adottata ha garantito un'ottimizzazione efficace e sistematica.

3.3.4 Overfitting nel Random Forest e risultati ottenuti

Per valutare la presenza di overfitting, sono state confrontate le metriche di errore tra il training set e il test set.

La valutazione sul Test Set è la seguente:

- **MAE**: 54.08
- **RMSE**: 75.09

Mentre la valutazione sul Training Set è la seguente:

- **MAE**: 36.50
- **RMSE**: 51.12

Si osserva che il modello presenta un certo grado di overfitting, dato che l'errore sul training set è significativamente inferiore rispetto a quello sul test set.
Questo suggerisce che il modello si adatta molto bene ai dati di training ma potrebbe avere una generalizzazione limitata.

Ci sono stati diversi tentativi di migliorare le prestazioni di questo modello, cambiano anche i valori degli iperparametri; il modello migliorava ma non performava abbastanza bene, c'era sempre un piccolo tasso di overfitting.

3.4 Gradient Boosting

Il Gradient Boosting è il terzo modello appreso; è un potente metodo di apprendimento supervisionato che appartiene alla famiglia degli ensemble learning, nello specifico ai metodi di boosting.
L'idea principale del Gradient Boosting è combinare più modelli deboli (di solito alberi decisionali) in un modello forte.
A differenza del Random Forest, che costruisce alberi in parallelo, il Gradient Boosting costruisce gli alberi sequenzialmente, in modo che ogni nuovo albero corregga gli errori commessi dai precedenti.

3.4.1 Iperparametri di Gradient Boosting

Gli iperparametri chiave di Gradient Boosting includono:

- **n_estimators**: il numero di alberi nella sequenza. Un numero maggiore può migliorare le prestazioni ma aumenta il rischio di overfitting;
- **max_depth**: la profondità massima degli alberi. Una profondità maggiore aumenta la capacità del modello ma può portare a overfitting;
- **learning_rate**: determina il contributo di ogni albero alla predizione finale. Valori più bassi richiedono più alberi per ottenere buone prestazioni;
- **subsample**: la frazione di dati utilizzata per ogni iterazione, introducendo una forma di regolarizzazione simile al bagging;
- **min_samples_split**: il numero minimo di campioni richiesti per suddividere un nodo. Valori più alti aiutano a ridurre l'overfitting.

3.4.2 Ottimizzazione degli Iperparametri con Grid Search e Cross Validation

Per trovare la migliore combinazione di iperparametri, anche in questo caso è stata utilizzata una **Grid Search** con **Cross Validation**.
Il procedimento è il medesimo del modello precedente; anche in questo caso è stata utilizzata una k-fold=5.

```
'gradient_boosting': {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 10],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 1.0],
    'min_samples_split': [2, 5, 10]
}
```

3.4.3 Creazione della tabella dei risultati

Viene creata una tabella delle combinazioni degli iperparametri, la quale viene salvata all'interno del file `iperparametri_gradient_boosting.csv`, all'interno della cartella `iperparametri` e nella sottocartella `tabelle`.
Di seguito una parte della tabella.

apprendimento_supervisionato > iperparametri > tabelle > iperparametri_gradient_boosting.csv

```
1  mean_fit_time,std_fit_time,mean_score_time,std_score_time,param_learning_rate,param_max_depth,param_min_samples_split,
   param_n_estimators,param_subsample,params,split0_test_score,split1_test_score,split2_test_score,split3_test_score,split4_test_score,
   mean_test_score,std_test_score,rank_test_score
2  0.3307563781738281 4,0.006623050916486675,0.004940986633300781,0.004652289954872928,0.01,3,2,100,0.8,"{'learning_rate': 0.01,
   'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 100, 'subsample': 0.8}",-106.0642208000661,-92.84854464579351,-111.
   33567623705919,-101.16942551664279,-96.90830731216903,-101.66523490234611,6.534167531699744,157
3  0.3252508640289306 6,0.013468266396970716,0.004155731201171875,0.005089710475517262,0.01,3,2,100,1.0,"{'learning_rate': 0.01,
   'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 100, 'subsample': 1.0}",-106.08963756936413,-92.80634231041351,-112.
   01076016315197,-101.2883090772874,-98.420747177285,-102.1231592595004,6.55294605392993,160
4  0.6563230514526367,0.0126801474315431,0.004163932800292969,0.0035848399950906005,0.01,3,2,200,0.8,"{'learning_rate': 0.01,
   'max_depth': 3, 'min_samples_split': 2, 'n_estimators': 200, 'subsample': 0.8}",-62.85161018098784,-55.12601220631057,-65.
   175057426700 3,-59.61876457531807,-60.33082496361183,-60.62045387058572,3.3763479555564238,140
```

Invece, la combinazione più performante è stata salvata nel file `iperparametri_gradient_boosting.json`, all'interno della cartella `iperparametri` e nella sottocartella `migliori`.
Di seguito vi è riportata la combinazione.

apprendimento_supervisionato > iperparametri > migliori > {} iperparametri_gradient_boosting.json > ...

```
1  {"learning_rate": 0.1, "max_depth": 3, "min_samples_split": 5, "n_estimators": 300, "subsample": 0.8}
```

3.4.4 Valutazione del modello

Anche in questo caso sono state utilizzate sempre le stesse metriche.

Per la valutazione del **Test Set**:

- **MAE**: 10.40

- **RMSE**: 13.86

Invece, la valutazione del **Training Set**:

- **MAE**: 4.14
- **RMSE**: 5.25

3.4.5 Risultati ottenuti

Il Gradient Boosting ha un errore molto più basso rispetto agli altri modelli sia sul training set che sul test set.
Le sue prestazioni sono eccellenti, con una notevole capacità di generalizzazione, visto che l'errore sul training set è più basso rispetto al training set. Questo suggerisce che non c'è overfitting e che il modello sta imparando a generalizzare bene sui dati non visti.

3.5 Grafici

Durante le valutazioni dei vari modelli, sono stati generati vari grafici per analizzare le prestazioni del modello e visualizzare i **valori** delle **metriche**.
Viene mostrato un grafico a barre che mostra le metriche di valutazione del modello, come l'errore assoluto medio (MAE)e la radice dell'errore quadratico medio (RMSE).

Questi grafici forniscono una *visione chiara* e immediata delle performance del modello, rendendo facile identificare punti di forza.

Grafico della performance sui dati di test:

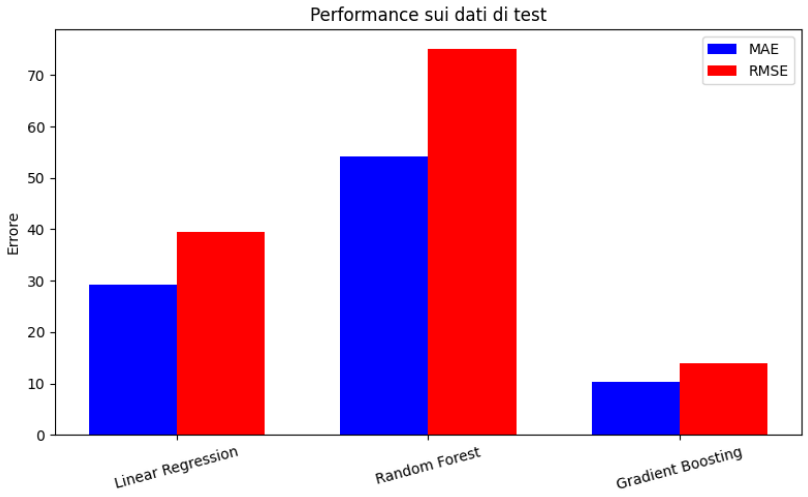
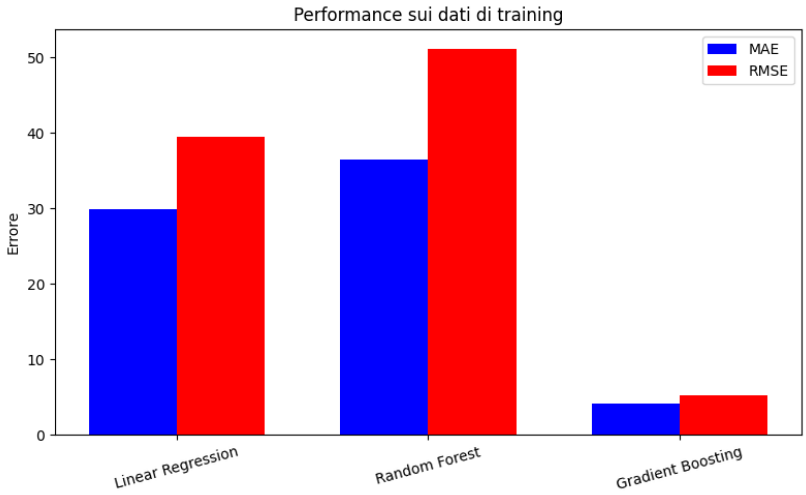


Grafico sulla performance sui dati di training:



Grazie alla visualizzazione delle metriche e delle predizioni, è possibile interpretare meglio i risultati e prendere decisioni informate sul miglioramento del modello.

I grafici sono salvati all'interno del progetto, nella cartella `grafici`.
I nomi rispettivi sono `test_set_metriche.png` e `training_set_metriche.png`.

3.6 Stima Calorie Bruciate

Dopo aver implementato i tre modelli di apprendimento supervisionato, si può eseguire il file `predict_models.py`.

```
Inserisci il genere (0 = Maschio, 1 = Femmina): 1

Seleziona il tipo di allenamento tra i seguenti:
0: Cardio
1: HIIT
2: Strength
3: Yoga
Inserisci il codice del tipo di allenamento: 2
Inserisci la durata della sessione in ore: 1.5
Inserisci il peso in kg: 45.7
Inserisci l'età: 23
Inserisci la media dei battiti per minuto a fine allenamento: 158

Risultati delle predizioni:
Regressione Lineare: 1171.22 kcal
Random Forest: 1091.84 kcal
Gradient Boosting: 1169.56 kcal
```

Si dovranno inserire i campi richiesti e verranno mostrati i valori delle calorie bruciate predetti da ogni modello.

4 Apprendimento Probabilistico

Il modello utilizzato nel progetto è basato sulla **Bayesian Ridge Regression**, una tecnica di **regressione probabilistica** che estende la regressione lineare introducendo un approccio bayesiano nella stima dei parametri del modello.

A differenza della regressione lineare tradizionale, che stima i coefficienti del modello utilizzando una singola soluzione deterministica, la Bayesian Ridge Regression assume una distribuzione a priori sui coefficienti, permettendo di ottenere una *distribuzione posteriore* che rappresenta l'*incertezza* sui parametri stimati.

Una volta addestrato, il modello viene utilizzato per predire il target, che nel nostro caso è il numero di calorie bruciate durante una sessione di allenamento, a partire da variabili come età, peso, tipo di allenamento e frequenza cardiaca media.

4.1 Funzionamento Regressione Bayesiana

La regressione bayesiana parte dal classico modello di regressione lineare:

$$Y = X\beta + \epsilon$$

dove:

- y è il valore target (calore bruciate nel nostro caso);
- X è la matrice delle feature (genere, tipo di allenamento...);
- β è il vettore dei parametri del modello (coefficienti da stimare);
- ϵ è il termine di errore (rumore nei dati).

Al modello base, si aggiunge un approccio bayesiano.
Invece di stimare un **unico valore** per ogni coefficiente β, la regressione bayesiana assegna una **distribuzione di probabilità** ai coefficienti.
Questa probabilità viene aggiornata con nuovi dati utilizzando il **teorema di Bayes**:

$$P(\beta|D) = \frac{P(D|\beta)P(\beta)}{P(D)}$$

dove:

- $P(\beta|D)$ è la distribuzione a posteriori dei parametri β dopo aver visto i dati D;
- $P(D|\beta)$ è la verosomiglianza (quanto bene il modello spiega i dati);
- $P(\beta)$ è la distribuzione a priori (conoscenza iniziale dei parametri);
- $P(D)$ è la probabilità dei dati osservati (costante di normalizzazione).

4.2 Iperparametri con Grid Search e Cross Validation

Nella fase di addestramento del modello, per migliorare le prestazioni del modello di Bayesian Ridge Regression, è stata applicata una tecnica di **Grid Search** combinata con la **Cross Validation** (CV).

Ricordiamo che la Grid Search è una tecnica che esplora in modo esaustivo una serie di possibili valori per ciascun iperparametro, al fine di trovare la combinazione che minimizza l'errore del modello.

Gli iperparametri utilizzati sono:

- **alpha_1**: parametro di regolarizzazione per la distribuzione a priori sui coefficienti. Controlla la varianza dei coefficienti nel modello;
- **alpha_2**: simile a **alpha_1**, è un altro parametro di regolarizzazione che agisce sui coefficienti del modello;
- **lambda_1**: ulteriore parametro di regolarizzazione che aiuta a bilanciare la complessità del modello e l'errore;
- **lambda_2**: come **lambda_1**, è un altro parametro di regolarizzazione che agisce sul modello;
- **fit_intercept**: determina se includere o meno un termine di intercetta nel modello. Le opzioni possibili sono:
  - `True`: includere il termine di intercetta;
  - `False`: non includere il termine di intercetta.

In questo caso, è stato utilizzato un **5-fold Cross Validation**, che significa che i dati sono stati suddivisi in 5 sottogruppi (fold), e per ogni combinazione di iperparametri, il modello è stato addestrato su 4 fold e testato sull'1 rimanente, ripetendo questa operazione per ogni fold.

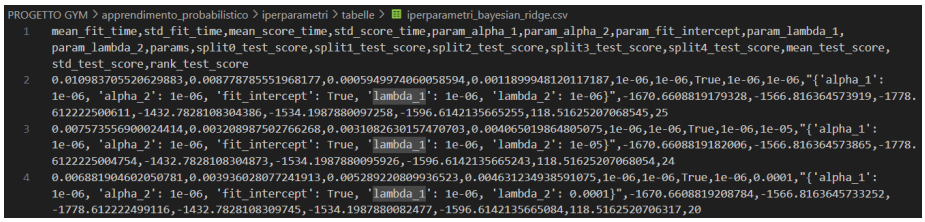
In questo modo, il modello è stato valutato su diverse porzioni dei dati, garantendo una stima più robusta delle sue prestazioni.
Al termine della Grid Search, i migliori parametri sono stati selezionati e utilizzati per addestrare il modello finale, che è stato successivamente valutato.

```
param_grid = {
    'alpha_1': [1e-6, 1e-5, 1e-4],
    'alpha_2': [1e-6, 1e-5, 1e-4],
    'lambda_1': [1e-6, 1e-5, 1e-4],
    'lambda_2': [1e-6, 1e-5, 1e-4],
    'fit_intercept': [True, False]
}
```

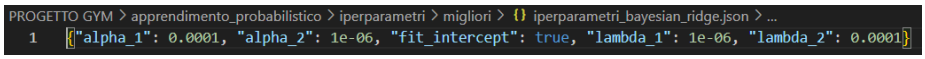
4.3 Creazione della tabella dei risultati

Anche in questo caso, una volta completata la Grid Search, i risultati ottenuti vengono salvati in una tabella, che mostra le **combinazioni di iperparametri** testate e il punteggio ottenuto per ciascuna.

La tabella è stata salvata nella cartella `iperparametri`, nella sottocartella `tabelle` ed il file si chiama `iperparametri_bayesian_ridge.csv` che si trova all'interno del progetto.
Di seguito vediamo una parte della tabelle presente nel file precedente:



Invece, i **migliori iperparametri** trovati (combinazione più performante) vengono salvati in un file JSON denominato `iperparametri_bayesian_ridge.json` nella cartella `iperparametri`, nella sottocartella `migliori`.



4.4 Intervallo di confidenza

Il modello Bayesian Ridge Regression utilizza una formulazione bayesiana della regressione lineare, il che significa che i parametri del modello non sono stimati come valori fissi, ma piuttosto come distribuzioni di probabilità.
Nell'apprendimento probabilistico, l'intervallo di confidenza viene calcolato per quantificare l'**incertezza** nelle previsioni del modello.

L'intervallo di confidenza viene calcolato come:

$$\text{Intervallo di Confidenza} = [\mu - 1.96\sigma, \mu + 1.96\sigma]$$

dove:

- μ è la media della previsione;
- σ è la deviazione standard della previsione;
- **1.96** è il valore della distribuzione normale standard corrispondente a un intervallo di confidenza del **95%**.

L'intervallo di confidenza fornisce una stima dell'incertezza attorno alla previsione.
Ad esempio, se il risultato è [200, 250] kcal, significa che, con il 95% di probabilità, il vero valore dell'output si troverà in quell'intervallo.

4.5 Valutazione del modello

Per valutare le prestazioni del modello di regressione bayesiana, sono state utilizzate diverse metriche di errore e di bontà di adattamento.
Le metriche scelte sono

- **Root Mean Squared Error (RMSE)**
- **Mean Absolute Error (MAE)**
- **R-squared (R^2)**

Queste metriche forniscono informazioni cruciali sulla precisione delle predizioni e sul grado di adattamento del modello ai dati.

Root Mean Squared Error (RMSE)

Il **Root Mean Squared Error (RMSE)** è una metrica che misura la deviazione quadratica media tra i valori predetti dal modello e i valori reali.
Viene calcolato come la radice quadrata della media dei quadrati degli errori (le differenze tra i valori predetti e quelli reali).
Un RMSE più basso indica un miglior adattamento del modello ai dati.

Nel nostro caso, i valori di RMSE ottenuti sono:

- **Train RMSE**: 39.49
- **Test RMSE**: 39.47

I valori di RMSE sono molto simili tra il training set e il test set, indicando che il modello non ha subito overfitting e ha generalizzato bene sui dati non visti.

Mean Absolute Error (MAE)

Il **Mean Absolute Error (MAE)** misura l'errore medio assoluto tra le predizioni del modello e i valori reali. È una metrica che non penalizza eccessivamente gli errori più grandi, ma fornisce comunque una valutazione diretta della grandezza degli errori.
Un MAE basso indica che, mediamente, le predizioni del modello sono vicine ai valori reali.

I valori di MAE ottenuti sono:

- **Train MAE**: 29.90

- **Test MAE**: 29.31

Anche in questo caso, i valori di MAE per il training e il test set sono simili, suggerendo che il modello è in grado di fare previsioni accurate su nuovi dati.

R-squared (R^2)

Il **R-squared (R^2)** è una misura della **bontà** di adattamento del modello, che rappresenta la proporzione della varianza dei dati che è spiegata dal modello.
Il valore di $\mathbf{R^2}$ varia tra 0 e 1, con valori più alti che indicano un modello che spiega una maggiore parte della variabilità nei dati. Un valore di $\mathbf{R^2}$ prossimo a 1 indica una buona capacità predittiva del modello.

I valori di R^2 ottenuti sono:

- **Train** R^2: 0.9783
- **Test** R^2: 0.9813

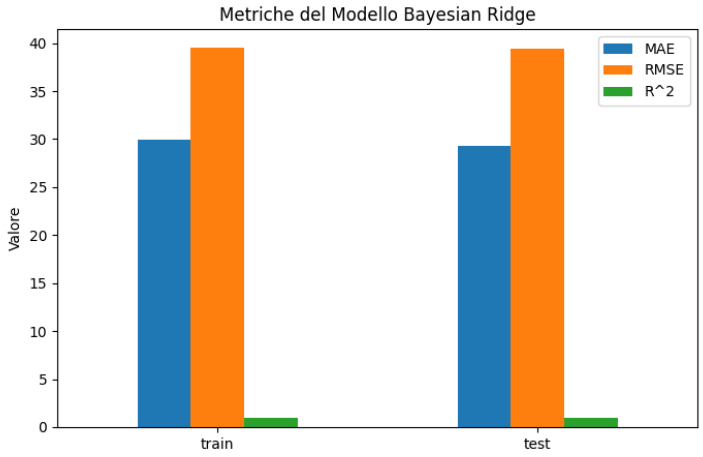
Entrambi i valori di R^2 sono molto elevati, indicando che il modello è in grado di spiegare quasi tutta la variabilità dei dati, sia nel training set che nel test set. In particolare, il modello sembra avere una performance eccellente anche sui dati non visti.

Nel complesso, le metriche ottenute suggeriscono che il modello di regressione bayesiana ha *performance eccellenti*. Le similitudini tra i risultati ottenuti sul training set e sul test set indicano che il modello non ha subito overfitting e ha generalizzato bene ai nuovi dati.

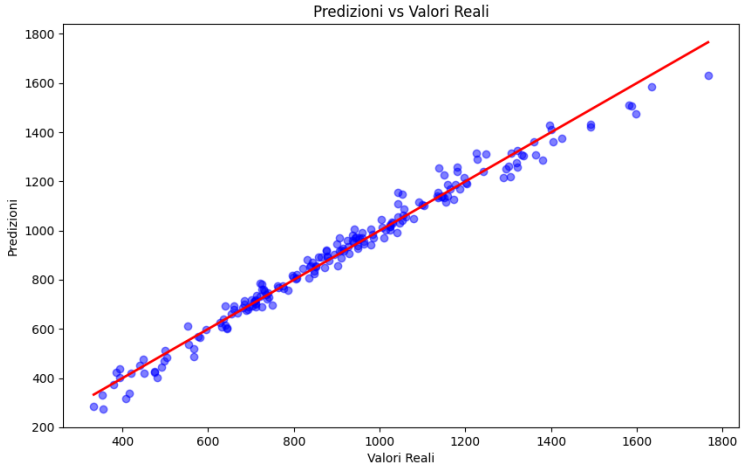
4.6 Grafici

Dopo aver calcolato le metriche, il codice genera alcuni grafici per avere una visione più chiara.
I grafici sono salvati all'interno della cartella `grafici`.

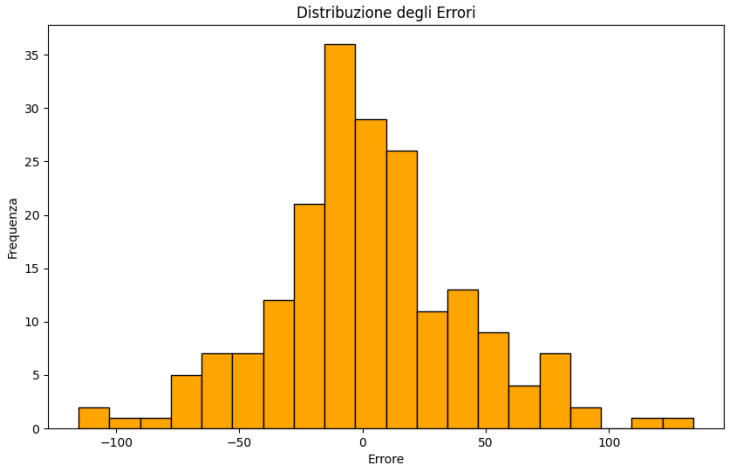
Il grafico `metriche_bayesian_ridge.png` è un grafico che riporta i valori delle metriche calcolate sia sul test che sul training set.
Questo grafico a barre mostra le metriche di valutazione per il modello, inclusi il **MAE**, il **RMSE** e l'$\mathbf{R^2}$ sia per il training che per il test set. Le barre permettono di confrontare visivamente le prestazioni del modello sui dati di addestramento e di test.



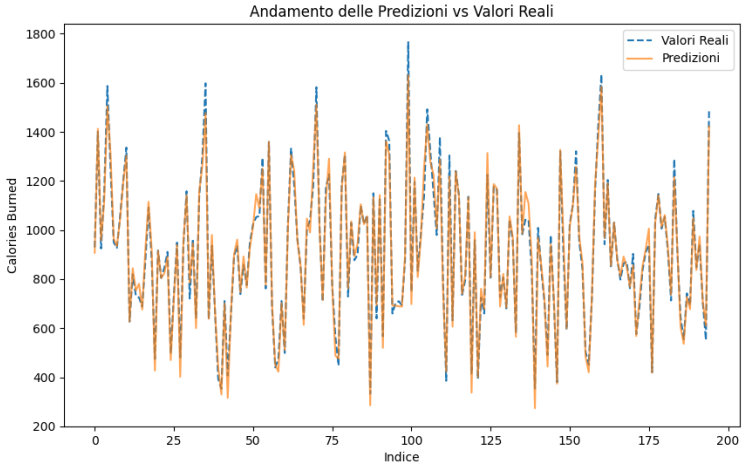
All'interno del grafico `predizioni_vs_valori_reali.png` vengono confrontati i valori reali delle calorie bruciate con le predizioni del modello.
Una linea rossa rappresenta la "linea di perfetta corrispondenza", ossia dove le predizioni e i valori reali sarebbero identici.
Questo grafico aiuta a visualizzare quanto bene il modello predice i valori reali.



Il grafico `distribuzione_errori.png` mostra la distribuzione degli errori (differenza tra i valori reali e le predizioni) sui dati di test.
La forma dell'istogramma fornisce informazioni sulla distribuzione degli errori del modello, che idealmente dovrebbe essere centrata attorno a zero per indicare errori equamente distribuiti.



Infine, il grafico `confronto_predizioni_valori_reali` mostra l'andamento delle predizioni rispetto ai valori reali delle calorie bruciate.
La linea tratteggiata rappresenta i valori reali, mentre la linea continua rappresenta le predizioni del modello. Se le linee si sovrappongono bene, significa che il modello sta facendo previsioni accurate.



4.7 Stime Calorie Bruciate

Ecco un esempio con l'apprendimento probabilistico:

```
=== Risultati delle Predizioni ===
Regressione Lineare: 1164.24 kcal
Random Forest: 1114.76 kcal
Gradient Boosting: 1154.80 kcal
Bayesian Ridge: 1164.19 kcal
Intervallo di Confidenza Bayesian Ridge: [1086.19, 1242.19] kcal
```

4.8 Modello SGDClassifier fallimentare

Prima di implementare il modello di Regressione Lineare Bayesiana, ho implementato il modello **SGDClassifier**. Esso è un classificatore che utilizza l'algoritmo *Stochastic Gradient Descent* (SGD), che è una versione stocastica (ovvero che aggiorna i parametri ad ogni singolo campione) del Gradient Descent.

All'inizio, il modello inizia con parametri (pesature dei vari feature) casuali o con valori molto piccoli.
Per ogni dato di addestramento, l'algoritmo calcola il gradiente della funzione di errore (o perdita) rispetto ai parametri. Ad esempio, nel caso della regressione logistica, la funzione di perdita utilizzata è il *log-loss*, che misura quanto l'output predetto dal modello si discosta dall'etichetta vera.
L'algoritmo aggiorna i parametri (pesi) del modello utilizzando la regola del gradiente discendente: si spostano nella direzione opposta al gradiente, per ridurre l'errore. Poiché l'algoritmo è stocastico, aggiorna i parametri dopo ogni singolo esempio di addestramento invece che su tutto il batch.

Durante l'implementazione, sono emerse alcune problematiche:

- **Classi sbilanciate**: sebbene il dataset originale fosse sbilanciato, è stato applicato il metodo **SMOTE** (Synthetic Minority Over-sampling Technique) per bilanciare le classi.
  Questo ha migliorato la distribuzione delle etichette nel set di addestramento, ma il modello SGDClassifier non è riuscito a generalizzare bene, come dimostrato dall'accuratezza del test set.
  Nonostante la bilanciatura, il modello non ha raggiunto un'accuratezza soddisfacente, fermandosi al 75.9%, che è un risultato inferiore a quello che ci si aspettava per un problema di classificazione relativamente semplice;

- **Alta varianza nei parametri**: durante la fase di Grid Search per la ricerca dei migliori iperparametri, la ricerca ha esplorato una vasta gamma di valori per alpha, learning_rate, eta0 e max_iter.
  Sebbene siano stati testati diversi set di parametri, nonostante la ricerca, il miglior modello trovato non ha migliorato significativamente l'accuratezza del modello, suggerendo che il modello SGDClassifier potrebbe non essere adatto per il tipo di distribuzione dei dati o per il bilanciamento delle classi;

- **Complessità del modello**: il modello SGDClassifier è più sensibile al tipo di dati e alla loro distribuzione rispetto ad altri approcci probabilistici. In particolare, la regressione logistica (impiegata tramite loss='log_loss') è una tecnica che può soffrire in presenza di variabili altamente correlate o di un bilanciamento non perfetto delle classi.

4.8.1 Conclusione

In conclusione, nonostante gli sforzi di ottimizzazione con il SGDClassifier e l'utilizzo di SMOTE per bilanciare le classi, il modello non ha raggiunto un'accuratezza soddisfacente.
Questo ha portato alla decisione di scartare l'approccio di apprendimento per gradiente stocastico a favore di un modello probabilistico, in particolare la regressione bayesiana, che ha mostrato performance migliori e una gestione più robusta della probabilità di appartenenza alle classi.

5 Rappresentazione della Conoscenza - KB

La Base di Conoscenza (KB) è un componente fondamentale del sistema, progettata per fornire raccomandazioni personalizzate basate su regole logiche. Utilizzando `Prolog`, la KB permette di inferire il tipo di allenamento più adatto, l'intensità consigliata e la durata ottimale della sessione, in base alle caratteristiche dell'utente.

L'obiettivo principale della KB è quello di sfruttare una rappresentazione formale della conoscenza per:

- determinare il tipo di allenamento più adatto in base all'**indice di massa corporea** (BMI);
- stimare il **dispendio calorico** in base al tipo di workout, alla durata e al peso corporeo;
- assegnare un **livello di intensità** in base alle calorie bruciate;
- suggerire una **durata ottimale** dell'allenamento, adattata al livello di intensità.

L'inferenza viene eseguita tramite **regole logiche** definite in Prolog, che combinano **fatti predefiniti** e **regole** derivate per generare raccomandazioni basate su dati personalizzati.

5.1 Struttura della KB

La Base di Conoscenza è composta da due elementi principali:

- **Fatti**: rappresentano informazioni di base predefinite, come il consumo calorico per tipo di workout o le soglie di intensità;
- **Regole**: definiscono la logica inferenziale utilizzata per determinare il workout più adatto, stimare le calorie bruciate e suggerire la durata ottimale dell'allenamento.

5.1.1 Fatti

I fatti nella KB rappresentano conoscenze statiche che il sistema utilizza per effettuare le inferenze. Alcuni esempi di fatti definiti sono:

- **Dispendio calorico per tipo di workout**: ogni tipo di allenamento ha un valore fisso di consumo calorico per kg di peso corporeo per ora:

```
calories_per_kg(0, 8.0).   % Cardio
calories_per_kg(1, 10.0).  % HIIT
calories_per_kg(2, 6.0).   % Strength Training
calories_per_kg(3, 3.5).   % Yoga
```

- **Soglie di intensità**: le soglie di intensità vengono utilizzate per classificare il livello di allenamento in base alle calorie bruciate:

```
intensity_threshold(high, 800).
intensity_threshold(moderate, 400).
```

- **Assegnazione del workout consigliato**: a seconda del livello di fitness, viene assegnato un tipo di allenamento consigliato:

```
workout_type(underweight, 2). % Strength Training
workout_type(normal, 0). % Cardio
workout_type(overweight, 1). % HIIT
workout_type(obese, 3). % Yoga
```

- **Durata consigliata dell'allenamento**: la durata suggerita di un workout dipende dal livello di intensità:

```
suggested_duration(high, 0).
suggested_duration(moderate, 0.5).
suggested_duration(low, 1).
```

5.1.2 Regole

Le regole della KB definiscono il modo in cui i fatti vengono combinati per generare raccomandazioni personalizzate.
Alcune delle principali regole implementate sono:

- **Calcolo del BMI (Indice di Massa Corporea)**: il BMI viene calcolato come:

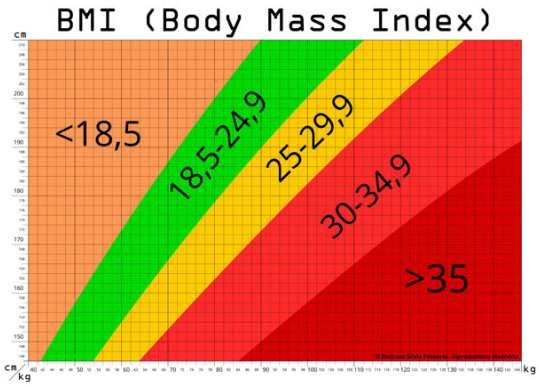
$$BMI = \frac{\text{Peso}}{\text{Altezza}^2}$$

In Prolog:

```
bmi(Weight, Height, BMI) :-
Height > 0,
BMI is Weight / (Height * Height).
```

- **Determinazione del livello di fitness**: in base al BMI calcolato, il sistema assegna un livello di fitness:

```
fitness_level(BMI, underweight) :- BMI < 18.5.
fitness_level(BMI, normal) :- BMI >= 18.5, BMI =< 24.9.
fitness_level(BMI, overweight) :- BMI >= 25, BMI =< 29.9.
fitness_level(BMI, obese) :- BMI >= 30.
```



- **Classificazione dell'intensità dell'allenamento**: basata sulle calorie bruciate:

```
intensity_category(Calories, high) :-
intensity_threshold(high, Threshold),
Calories > Threshold.
intensity_category(Calories, moderate) :-
intensity_threshold(high, HighThreshold),
intensity_threshold(moderate, ModerateThreshold),
Calories =< HighThreshold,
Calories >= ModerateThreshold.
intensity_category(Calories, low) :-
intensity_threshold(moderate, Threshold),
Calories < Threshold.
```

- **Determinazione dell'intensità raccomandata**: determina l'intensità consigliata di un allenamento in base al peso, all'altezza e alla durata dell'esercizio.

```
recommended_intensity(Weight, Height, Intensity, Duration) :-
recommended_workout(Weight, Height, Workout_Type),
calories_burned(Workout_Type, Duration, Weight, Calories),
intensity_category(Calories, Intensity).
```

5.2 Ragionamento Inferenziale nella KB

La base di conoscenza (KB) non rappresenta semplicemente un insieme di fatti statici, ma una rete di regole logiche che consentono un **ragionamento inferenziale**.
In questo contesto, l'inferenza non avviene mediante il recupero diretto di informazioni predefinite, ma attraverso il collegamento dinamico di fatti e regole.

Ogni regola può essere vista come un modulo computazionale che, a partire da un insieme di input, produce delle conclusioni, potenzialmente utilizzate da altre regole.

Il sistema, quindi, non restituisce una risposta immediata alla query, ma costruisce la risposta tramite una **sequenza di inferenze**, dove ogni passo del processo dipende dal risultato del passo precedente.
Questo approccio inferenziale è tipico dei sistemi basati su *logica del primo ordine*, come quelli implementati tramite il linguaggio Prolog, dove ogni inferenza è il risultato dell'applicazione di regole che operano sui dati esistenti.

La propagazione delle informazioni non avviene istantaneamente: una query può generare una sequenza di interrogazioni che, tramite l'applicazione di regole logiche, trasformano i dati iniziali in conclusioni più complesse.
In altre parole, il sistema non estrae semplicemente fatti dalla base di conoscenza, ma li deriva applicando le regole definite nella KB.

Esempio Per esempio, se si richiede una predizione sul tipo di allenamento consigliato, il sistema non recupera semplicemente il tipo di allenamento da un fatto predefinito, ma deve calcolare il BMI dell'utente utilizzando il suo peso e la sua altezza.
Successivamente, sulla base del BMI, viene determinato il livello di fitness dell'utente (ad esempio, "sovrappeso").
Questo livello di fitness, quindi, si collega al tipo di allenamento più adatto (ad esempio, un allenamento ad alta intensità, come l'HIIT per una persona sovrappeso).

5.3 Considerazioni finali

Il sistema basato sulla base di conoscenza (KB) per il calcolo della spesa calorica e la raccomandazione di allenamenti ha dimostrato di essere un'implementazione robusta e utile per la gestione personalizzata delle sessioni di allenamento. Tuttavia, come ogni sistema complesso, ci sono diverse direzioni in cui il progetto potrebbe evolvere per migliorare l'accuratezza, l'usabilità e la generalità delle inferenze.

Ad esempio si può procedere con l'espansione del modello di allenamento. Attualmente, il sistema copre allenamenti di tipo cardio, HIIT, forza e yoga.
Tuttavia, si potrebbe ampliare l'offerta includendo nuove categorie di allenamento, come il pilates o l'allenamento funzionale, che potrebbero meglio adattarsi alle preferenze o alle necessità degli utenti. L'inclusione di nuovi tipi di workout, accompagnata da una ricerca più approfondita sui parametri di dispendio calorico, potrebbe rendere il sistema più versatile e completo.

Un ulteriore sviluppo importante potrebbe essere l'espansione della base di

conoscenza, includendo variabili personalizzate, come lo storico degli allenamenti, i livelli di recupero tra le sessioni e parametri psicologici legati alla motivazione e allo stress.

5.4 Esempio pratico

Osserviamo un esempio pratico.

```
Inserisci il genere (0 = Maschio, 1 = Femmina): 0

Seleziona il tipo di allenamento tra i seguenti:
0: Cardio
1: HIIT
2: Strength
3: Yoga
Inserisci il codice del tipo di allenamento: 0
Inserisci la durata della sessione in ore: 0.50
Inserisci il peso in kg: 73
Inserisci l'altezza in metri: 1.80
Inserisci l'età: 23
Inserisci la media dei battiti per minuto a fine allenamento: 121

=== Risultati dell'Inferenza Prolog ===
Workout consigliato: Cardio
Intensità stimata: low
Durata ottimale: 1.5 ore

=== Risultati delle Predizioni ===
Regressione Lineare: 316.78 kcal
Random Forest: 488.45 kcal
Gradient Boosting: 356.95 kcal
Bayesian Ridge: 316.87 kcal
Intervallo di Confidenza Bayesian Ridge: [238.57, 395.17] kcal
```

6 Dipendenze

Le seguenti librerie devono essere installate per il corretto funzionamento del progetto:

- **pandas**: libreria per la manipolazione e l'analisi dei dati in modo efficiente;
- **joblib**: utilizzata per il salvataggio e il caricamento di oggetti Python;
- **scikit-learn**: libreria di machine learning che fornisce strumenti per la modellazione predittiva, tra cui algoritmi di regressione, classificazione, clustering e riduzione della dimensionalità;
- **numpy**: libreria per il calcolo numerico ad alte prestazioni, utilizzata per operazioni su array e calcolo di metriche;

- **matplotlib**: libreria per la creazione di grafici e visualizzazioni statiche in Python;
- **pyswip**: libreria che permette l'integrazione di Prolog in Python, utile per la gestione di una Knowledge Base (KB) e per eseguire inferenze logiche basate su regole;

6.1 Installazione delle Dipendenze

1. Installare Python da `https://www.python.org/`
2. Installare SWI-Prolog da `https://www.swi-prolog.org/`

Per installare tutte le librerie, eseguire il seguente comando:

```
pip install nomelibreria
```

All'interno della cartella del progetto c'è un file chiamato `requirements.txt` dove ci sono tutte le librerie da installare.
Si possono installare le librerie che servono digitando il seguente comando sul terminale:

```
pip install -r requirements.txt
```

7 Documentazione

All'interno del progetto ho inserito la documentazione sia di Python che di Prolog.

Documentazione in Python

La documentazione in Python si trova nella cartella `docs`.
Navigare nella sottocartella `_build` e aprire la cartella `html`.
All'interno di quest'ultima, si trova tutta la documentazione cercata che si può aprire utilizzando un browser web.

Documentazione in Prolog

La documentazione in Prolog si trova nella cartella `kb`, nella sottocartella `doc`.
All'interno di questa cartella, è possibile trovare il file principale `index.html` e il file `kb.html`, che possono essere aperti con un browser web per visualizzare la documentazione dettagliata.

8 Riferimenti Bibliografici

- `https://www.kaggle.com/`
- David L. Poole, Alan K. Mackworth - Artificial Intelligence_Foundations of Computational Agents-Cambridge University Press (2010)