Machine Learning Project

“Calories Estimation Problem”

2022/2023

Zaza Maria Elena

Perna Grazia

Index

1. Introduction
   1. Motivation
2. Dataset
3. Libraries
4. Preprocessing
5. Workflow
6. Extra Trees Regressor
7. Results
8. Conclusion
9. Bibliography

**Index**

[Introduction 4](#_Toc140943869)

[Motivation 4](#_Toc140943870)

[Dataset 5](#_Toc140943871)

[Libraries 7](#_Toc140943872)

[Pre-processing 8](#_Toc140943873)

[Workflow 9](#_Toc140943874)

[Results 9](#_Toc140943875)

[Extra Trees Regressor 11](#_Toc140943876)

# Introduction

Nowadays, the increasing prevalence of overweight and obesity has become a pressing public health concern. Sedentary lifestyles, unhealthy eating habits and lack of awareness about appropriate daily calorie intake contribute to this global issue.

For individuals seeking to maintain an healthy lifestyle, understanding their personalized calorie needs is crucial. However, calculating precise daily calorie requirements can be challenging, as it involves complex factors like metabolic rate, activity level, age and so on.

To address this problem, the project described in this paper, aims to develop a predictive model capable of estimating the right amount of daily calories an individual should consume. By considering multiple features like weight, height, Body Mass Index (BMI), Basal Metabolic Rate (BMR), age, gender and activity level, the model will provide personalized recommendations for daily calorie intake tailored to each person's unique needs. This will empower individuals to make informed dietary choices, promote healthier habits, and contribute to long-term weight management and overall well-being.

## Motivation

The first challenging part, was researching a suitable dataset for our task, as a matter of the fact that we met several difficulties to find a proper dataset that helped us to solve the problem proposed.

So, finally, for our study we selected a dataset that contained proper features which played an important role in determining an individual's calorie income.

To ensure the efficacy of our study, we strategically designed our research to explore various machine learning techniques, specifically focusing on ensemble models.

A reason why we followed this line was that ensemble methods use a "teamwork" approach to make predictions: they bring together several individual models, each having its own strengths and weaknesses, and instead of relying on just one model's judgment, they consider the opinions of all team members in order to reach a final decision by combining all the proposals.

The diversity between these different proposals, minimizes the risk of overfitting and improves generalization capability, making them well-suited for complex and high-dimensional data, like the one in our dataset.

Moreover, ensemble models are highly effective in handling noisy or incomplete data. Real-world datasets are often messy, noisy and contain various imperfections. By using ensemble techniques, we can enhance the robustness of our predictions and reduce the impact of errors in individual models.

Another significant advantage of ensemble models is their ability to capture complex relationships within the data, especially when the underlying patterns might be non-linear. The combination of different algorithms, such as Random Forest and Gradient Boosting, allows ensemble methods to take complex decision boundaries and accurately approximate the underlying data distribution.

# Dataset

Our dataset’s name is "reccomender.csv" and it contains the following features:

* Id (int): identification number for each individual;
* Age (integer), measured in years;
* Weight (real), measured in kilograms (kg);
* Height (real), measured in meters (m);
* Gender (categorical), with two possible values: 0 (male) or 1 (female);
* BMI (Body Mass Index) (real): a calculated value based on an individual's weight and height, providing an indication of their body composition;
* BMR (Basal Metabolic Rate) (real): represents the number of calories an individual needs to maintain their basic bodily functions at rest;
* Activity Level (real), ranging from 1 to 2, it quantifies the physical activity of an individual with higher values indicating a more active lifestyle;
* BMI\_tags (real);
* Label (integer).

We didn’t use the last two features as they were not so relevant for our task.

In the following image, there are some of the instances present in our dataset:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

According to the subdivision of our dataset for developing the model, we decided to split the dataset into three sets: the train set, the validation set and the test set.

We performed several data divisions to create different training, validation and test sets. After evaluating various percentages for each split, we found that the most effective division was to allocate 50% of the data to the training set, 20% to the validation set and 30% to the test set.

This split provided a balanced distribution of data for model training, tuning and final evaluation.

The reasons why we split the dataset in this way were:

* Having an huge portion of the data for training is crucial as it allows the models to learn the underlying patterns and relationships within the dataset.
* The validation set plays a relevant role in optimizing the hyperparameters of the models. It allows us to evaluate the models' performance during the training process without accessing the test set. By having a separate validation set, we can avoid overfitting to the test data and ensure that the models generalize well to new and unseen data.
* The test set, which is entirely independent of the training and validation data, has the primary purpose to assess the models' real-world performance and generalization ability.

To ensure a representative and unbiased split of the data, we have set the parameter ‘*shuffle*’ equals to True during the splitting process. This randomizes the order of the data before dividing it into the train, validation and test sets, preventing any potential bias due to the initial ordering of the data.

# Libraries

In our project, we used a set of powerful Python libraries to efficiently handle, preprocess and model the data.

The Pandas library provides us data structures and functions that enables to manipulate and analyze large datasets effectively. By using *train\_test\_split* from the sklearn.model\_selection module, we were able to split our dataset into distinct subsets for training, validation and testing.

To visualize our data and model results, we used Matplotlib library, which allowed us to create various plots and graphs. The sklearn.ensemble module offered several ensemble learning algorithms, including *RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor,* *IsolationForest, BaggingRegressor* and *ExtraTreesRegressor* for regression tasks.

To assess our models' performance, we employed various metrics such as *Mean Squared Error (MSE), R-squared* and *Mean Absolute Error (MAE)*, provided by the sklearn.metrics module. Additionally, we leveraged numpy, a fundamental numerical computing library, to perform essential mathematical operations.

As part of the data preprocessing pipeline, we used *StandardScaler, MinMaxScaler* and *RobustScaler*, from the sklearn.preprocessing module, that allowed us to standardize features, promoting the performance of certain algorithms.

To fine-tune our models and optimize their hyperparameters, we used *GridSearchCV* from the sklearn.model\_selection module, which helped us to make an exhaustive search over a specified parameter grid to identify the best hyperparameters for our models.

# Pre-processing

In our data preprocessing phase, we started by loading the dataset from a CSV file using the Pandas library. The feature we considered were age, weight, height, gender, BMI, BMR and activity level, and our target variable is "calories," representing the recommended daily calorie intake.

To prepare our data for machine learning algorithms, we experimented with three different scaling techniques to standardize the numerical features in our dataset. The objective was to ensure that all the features have similar scales and ranges, which is crucial for many machine learning algorithms to perform optimally.

First, we used the StandardScaler, which is a common scaling technique that standardizes features by removing the mean and scaling to unit variance. It works well when the features are normally distributed and there are no significant outliers.

Next, we tried the MinMaxScaler, which scales the features to a specified range, usually between 0 and 1. This scaler is suitable for cases where the data does not have a normal distribution and when we want to preserve the relative relationships between the features.

Finally, we tried also with the RobustScaler, which is robust to outliers and works well with data that contains extreme values or outliers. It scales the features by removing the median and scaling according to the interquartile range. This makes it less sensitive to extreme values and can be beneficial when the dataset contains significant outliers.

After applying each scaling technique, we proceeded with our modeling process and compared the performance of the different models with respect to the scaled features. We compared the different scaling techniques in order to find the most suitable one for our specific dataset and problem.

We found that the StandardScaler provided the best results for our model as it gave us the better performance.

# Workflow

After dividing our dataset into training, validation and test sets, we proceeded to experiment with several regression algorithms, specifically we used the following models:

* RandomForestRegressor
* AdaBoostRegressor
* IsolationForest (with auto-contamination parameter)
* ExtraTreesRegressor
* GradientBoostingRegressor

To ensure fair comparisons, before fitting each model, we applied the different scaling techniques we discussed about before.

Next, we evaluated the performance of these models using the validation set. This step was crucial in selecting the algorithm that provided the best results on unseen data. After comparing the different results, we found out that the ExtraTreesRegressor gave us the best results.

To further optimize the hyperparameters of the ExtraTreesRegressor, we employed the GridSearchCV, which performs an exhaustive search over specified parameter values to find the combination that yields the best performance.

We conducted the GridSearchCV with two different cross-validation strategies, k=5 and k=10, to assess the model's generalization performance. By using k=5 and k=10, we divided the training data into 5 and 10 subsets respectively, allowing us to evaluate the model's performance on multiple combinations of training and validation sets. This process provided us with a more reliable estimation of the model's ability to generalize to new and unseen data.

To confirm the model's true performance, we evaluated it on the previously unseen test set, providing a reliable estimation of its capabilities in real-world scenarios.

At the end, we found out that the model exhibited better generalization performance with k=10, indicating its robustness and suitability for real-world applications.

## Results

In the following table are represented the performance results of various regression models with different scalers applied to the dataset. The metrics used to evaluate the models are Mean Squared Error (MSE), R-squared (R2) and Mean Absolute Error (MAE).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MODEL | SCALER | MSE | R2 | MAE |
| Random Forest | Standard Scaler | 11.864398070945322 | 0.999916778658384 | 0.8483715772304115 |
| Random Forest | Robust Scaler | 11.74946587135911 | 0.999917584835974 | 0.8293156411451927 |
| Random Forest | MinMax Scaler | 12.239309779854953 | 0.9999141488869269 | 0.8510282170440043 |
| AdaBoost Regressor | Standard Scaler | 10516.91590916363 | 0.9262304040719556 | 88.57413587418999 |
| AdaBoost Regressor | Robust Scaler | 9979.27712149163 | 0.9300016043424882 | 87.19318572097343 |
| AdaBoost Regressor | MinMax Scaler | 10249.612991440541 | 0.9281053670745254 | 87.36538723769229 |
| Isolation Forest | Standard Scaler | 4666113.306158827 | -31.72987025120962 | 2126.9126747669775 |
| Isolation Forest | Robust Scaler | 4666158.365661622 | -31.730186315476246 | 2126.9193325565916 |
| Isolation Forest | MinMax Scaler | 4666650.773440051 | -31.733640248469555 | 2127.035178095872 |
| Extra Trees Regressor | Standard Scaler | 2.916498720201835 | 0.9999795425832086 | 0.5641549181091949 |
| Extra Trees Regressor | Robust Scaler | 3.2364421181235263 | 0.9999772983801182 | 0.5808764660452832 |
| Extra Trees Regressor | MinMax Scaler | 3.6857091616946556 | 0.9999741470524329 | 0.6285649067909375 |
| Gradient Boosting Regressor | Standard Scaler | 415.37980449357497 | 0.9970863701299103 | 15.569772441702446 |
| Gradient Boosting Regressor | Robust Scaler | 415.4127275852946 | 0.9970861391949866 | 15.57087714368953 |
| Gradient Boosting Regressor | MinMax Scaler | 415.31021142154356 | 0.9970868582818407 | 15.567141537607998 |
| Bagging Regressor  (estimator=ExtraTreesRegressor) | MinMax Scaler | 3.339583642736372 | 0.9999765749067483 | 0.6693924169774055 |

The obtained results of the different regression models with various scalers can be explained by considering the characteristics of each algorithm and how they interact with the scaling techniques.

The Random Forest Regressor and Extra Trees Regressor achieved exceptional performance in predicting calorie values. These ensemble methods work well for this task due to their ability to handle non-linear relationships and capture complex interactions between features. Additionally, their robustness against overfitting and their capacity to manage large feature spaces contribute to their exceptional performance.

On the other hand, the AdaBoost Regressor showed bad results: this could be due to AdaBoost's dependence on weak learners, which might not fully capture the complexity of the calorie estimation problem.

As regards the Isolation Forest Regressor, its negative R2 values indicate its inability to capture meaningful patterns in the data, suggesting that this particular algorithm is not well-suited for calorie estimation.

As for the Gradient Boosting Regressor, it showed better results compared to the two last model described earlier, but it was still less performant with respect to the Random Forest and Extra Trees regressors. Gradient Boosting can be effective at capturing complex relationships but it may require more fine-tuning and optimization to match the performance of the ensemble-based models.

Overall the Extra Trees regressor proved to be the best well-suited model for this specific regression task and delivered accurate calorie predictions, making it the top choice for calorie estimation in this study.

After finding out that the Extra Tree is the most performant regressor in this task, we plotted the distance between the actual prediction and the computed prediction after fitting the model.

Immagine che contiene testo, Diagramma, linea, schermata

Descrizione generata automaticamente

Then we used the GridSearchCV in order to find the best hyperparameters. We found the best hyperparameters both for cv=10 and cv=5, obtaining that:

* Best hyperparameters for cv=10: {'max\_depth': 35, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 720}
* Best hyperparameters for cv=5: {'max\_depth': 35, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 700}

In the following table the results obtained by using these two configurations are represented, showing that the most promising one is by setting the cv = 10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MODEL | MSE | MAE | R2 | CV |
| Extra Trees Regressor | 1.7879791700564407 | 0.4298720853080573 | 0.9999885946000565 | 10 |
| Extra Trees Regressor | 1.8558990653633354 | 0.4453104702124264 | 0.9999881613435717 | 5 |

## Extra Trees Regressor

The Extra Trees Regressor is an ensemble learning method used for regression tasks. It is an extension of the Random Forest algorithm and shares similarities with it. As a matter of the fact that, like Random Forest, the Extra Trees Regressor builds multiple decision trees and combines their predictions to obtain a more accurate and robust output.

The main difference between Extra Trees and Random Forest is the way they create individual trees. While Random Forest randomly selects a subset of features at each split, the Extra Trees Regressor takes this randomness a step further: it uses random subsets of features but it also chooses random thresholds for splitting the nodes in each decision tree. This extra level of randomness introduces additional diversification in the model, making it less prone to overfitting.

The main advantages of the Extra Trees Regressor are its simplicity, fast training speed and reduced variance (ability of a model to produce consistent and stable predictions when exposed to different subsets of the training data). By introducing more randomness during the tree-building process, it can explore various features and data splits, effectively reducing bias and making it less sensitive to noise in the training data.

Due to its ability to handle high-dimensional datasets, the Extra Trees Regressor is particularly useful when working with a large number of features or dealing with noisy data. It can be especially effective when there are several irrelevant or redundant features.

A disadvantage is that it may require more trees to achieve similar performance compared to a carefully tuned Random Forest. However, its fast training time and competitive performance often outweigh this concern, especially for large datasets.

# Conclusion