

NutriScore ML Predictor using OpenFoodFacts Data

Jacopo Parretti, Cesare Fraccaroli
Student IDs: VR536104, VR533061
MsC in Artificial Intelligence - Machine Learning course
Academic Year 2025-26

Contents

1	Motivation and Rationale	2
1.1	Contextual Framework	2
1.2	Problem Statement	2
1.3	Rationale	2
1.4	Significance	2
2	State of the Art (SOTA)	2
3	Objectives	3
4	Methodology	3
4.1	Loading and Filtering	3
4.2	Exploratory Data Analysis	3
4.3	Preprocessing	7
4.3.1	Missing Value Handling	8
4.3.2	Outlier Detection and Removal	8
4.3.3	Categorical Feature Encoding	8
4.3.4	Feature Engineering	9
4.3.5	Feature Scaling	10
4.3.6	Dimensionality Reduction	10
4.3.7	Data Splitting	10
4.4	Models	11
4.4.1	Logistic Regression	11
4.4.2	K-Nearest Neighbors	11
4.4.3	Support Vector Machine	11
4.4.4	Random Forest	11
4.4.5	XGBoost	11
5	Experiments & Results	12
5.1	Experimental Setup	12
5.2	Performance Metrics	12
5.3	Model-Specific Analysis	13
5.4	Cross-Validation Results	14
5.5	Error Analysis	14
5.6	Observations	14
6	Conclusions	15
7	References	16

1 Motivation and Rationale

1.1 Contextual Framework

Non-Communicable Diseases (NCDs) linked to modern diets are important nowadays. People need access to nutritional information, but interpreting nutritional tables is not so easy. Front-of-Pack labels like Nutri-Score help by giving an immediate visual for healthier choices. Predicting these grades accurately matters because today's food landscape is dominated by ultra-processed foods. The increased availability of these foods has changed how the world eats and drinks, with lots of negative effects on health [1].

1.2 Problem Statement

The task we are trying to solve is the prediction of the Nutri-Score grade given some data about food and culinary products. The dataset we are using is obtained from the Open Food Facts (<https://it.openfoodfacts.org/>) website. The food dataset (<https://static.openfoodfacts.org/data/en.openfoodfacts.org.products.csv.gz>) is a vast crowd-sourced dataset. It is large but prone to many errors, invalid and missing data, and different languages.

The Nutri-Score is usually calculated using a deterministic formula that uses fiber, fruits, protein, etc. However, all this data is not always present in the dataset. When not all of these nutrients are present, we are left with unclassified products.

1.3 Rationale

We want to use Machine Learning in order to solve the problem of unclassified products. Machine learning can act as an approximator capable of predicting the grade based on partial macronutrient data or pattern handling missing or wrong values. We want to try filling the gap in the dataset by automating the classification.

1.4 Significance

We, as MSc students, want to try investigating classification performance in a highly noisy, real-world dataset. The classification can maybe help the usage of the Open Food Facts dataset for third-party applications or for research.

2 State of the Art (SOTA)

Research on crowded nutritional databases like this shows two main problems: data sparsity and the non-linear nature of quality indices like Nutri-Score.

Traditional approaches used linear models, but these don't work well. They can't preserve macronutrient correlations or capture the thresholds in the Nutri-Score calculation. Logistic Regression has limitations because it can't model the decision rules that Nutri-Score uses.

Current state-of-the-art methods use multivariate imputation via KNN to keep the local data structure, combined with non-linear ensemble models. Ensemble models handle outliers well and can model complex relationships in imbalanced datasets [4].

In this project, we explore Random Forest and XGBoost as state-of-the-art ensemble models. Research suggests that better solutions exist for missing data imputation: Non-Negative Matrix Factorization (NMF), Multiple Imputations by Chained Equations (MICE), Nonparametric Missing Value Imputation using Random Forest (MissForest), and K-Nearest Neighbors (KNN). These methods perform better than simple approaches like using mean or median [2].

3 Objectives

The main objective of the project is to develop a robust machine learning framework capable of predicting the Nutri-Score grade (classes A through E) for products in the Open Food Facts database. The main problem is the issue of incomplete nutritional metadata, which in most cases, as we will see, prevents the deterministic calculation of the score for a significant portion of the samples. To achieve this, the project tries:

- **Algorithmic Benchmarking:** Conduct a comparative analysis between linear models (Logistic Regression), distance-based algorithms (KNN), non-linear methods (SVM with RBF kernel), and ensemble tree methods (Random Forest, XGBoost). The goal is to identify the best model for the approximation of the Nutri-Score.
- **Optimization for Class Imbalance:** To maximize the F1-Macro score rather than simple Accuracy as the primary success criterion. This ensures the model performs reliably across all nutritional grades.
- **Efficiency Analysis:** To evaluate the trade-off between predictive performance and computational cost (training time), identifying a model that is both accurate and scalable.

4 Methodology

The project uses and elaborates data from the Open Food Facts dataset. The dataset is open source and under the Open Database License. We use the dataset URL to download with a simple Python script.

4.1 Loading and Filtering

The total number of rows is over 4 million. Since the size is very large (over 9 GB), we decided to adopt a chunk-based solution: we first split into chunks and then open each chunk to filter the data. First, we delete the entries that do not have the target variable Nutri-Score grade. After this operation, we have around 1 million products. We decided to keep this set of columns:

```
'nutriscore_grade', 'code', 'product_name', 'brands', 'categories', 'countries',  
'energy_100g', 'energy-kcal_100g', 'fat_100g', 'saturated-fat_100g',  
'carbohydrates_100g', 'sugars_100g', 'fiber_100g', 'proteins_100g',  
'salt_100g', 'sodium_100g', 'fruits-vegetables-nuts_100g',  
'fruits-vegetables-nuts-estimate_100g', 'additives_n', 'ingredients_n',  
'nutrition_grade_fr', 'pnns_groups_1', 'pnns_groups_2', 'main_category'
```

After filtering, we proceed with a sampling of 250,000 products with a set random state to ensure reproducibility. After this, we performed an exploratory data analysis to check the data.

4.2 Exploratory Data Analysis

To better understand the dataset, an exploratory data analysis on the 250k samples is needed. The project focuses on the prediction of the target variable Nutri-Score grade. Nutri-Score Grade does not appear evenly across all possible values A, B, C, D, E. The random sampling we conducted should guarantee that this can represent well the distribution in all of the dataset.

Class imbalance:

- **A = 14.4%**
- **B = 11.6%** (least common)

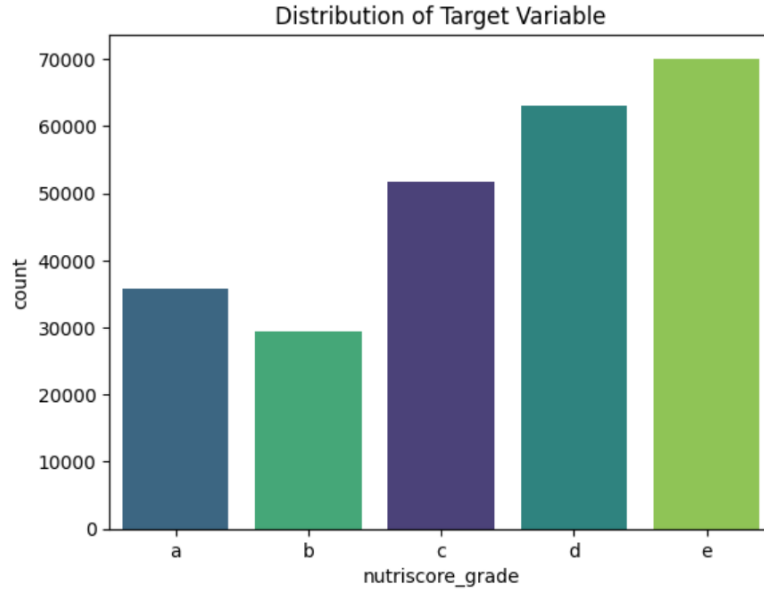


Figure 1: Distribution of the Nutri-Score grades (A–E) showing class imbalance.

- **C = 20.5%**
- **D = 25.2%**
- **E = 28.2%** (most common)

Then we analyze the missing percentages in the features we selected.

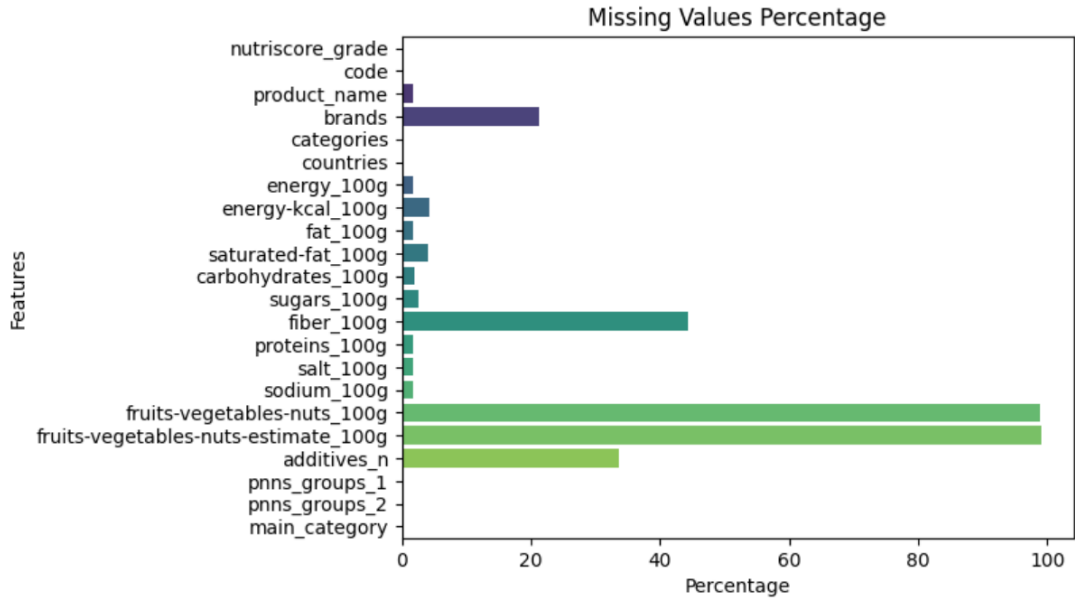


Figure 2: Percentages of missing values in the selected features.

It appears that `fruits-vegetables-nuts` and `fruits-vegetables-nuts-estimate` have more than **95 percent** of missing values.

We now try plotting the skewness of the other features. There are some features that have a high skew value. A skew value of over 300 means that there are some extreme outliers, maybe some invalid or out-of-range data that should be removed.

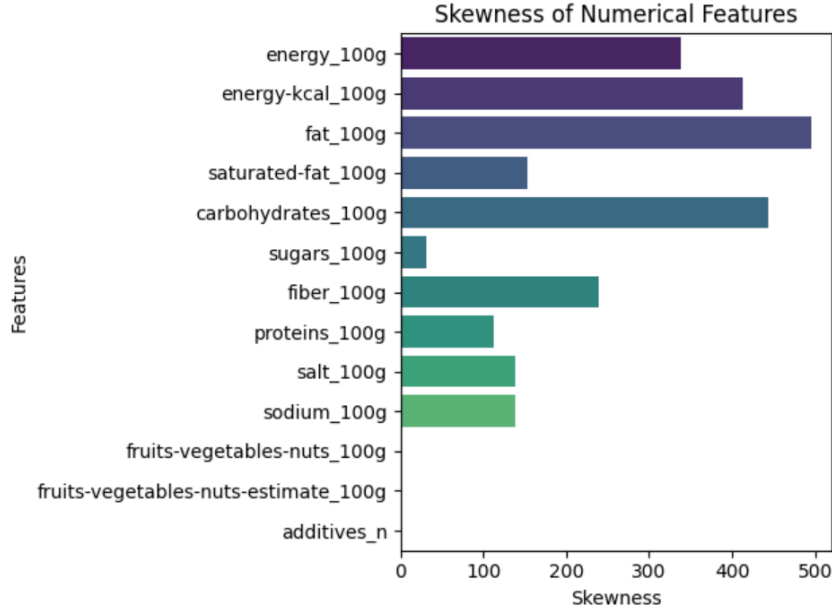


Figure 3: Skewness values of features before outlier removal. Extreme right-skewed values indicate strong zero-inflation and outliers.

energy_100g	134000.00
energy-kcal_100g	32100.00
fat_100g	90937.00
saturated-fat_100g	992276.00
carbohydrates_100g	8761109.00
sugars_100g	765433221122.00
fiber_100g	938364.00
proteins_100g	8578293.00
salt_100g	3625.00
sodium_100g	1450.00
fruits-vegetables-nuts_100g	100.00
fruits-vegetables-nuts-estimate_100g	100.00
additives_n	36.00

After a check on the values, we can clearly see some inconsistencies in the data.

Positive skewness is a structural characteristic of nutritional data. The distribution typically shows high near zero values (**zero-inflation**) and an elongated right tail.

In the nutritional domain, this pattern is common: the vast majority of products have minimal amounts of specific nutrients (e.g., **salt** or **fiber**), while only a small part of specific categories (e.g., **concentrated sauces** or **whole-grain cereals**) possess high concentrations.

Now we plot each feature singularly against the target variable to get some insights. Some features like fat, energy, sugars, and saturated fat clearly impact negatively on the Nutri-Score. The ones that seem to have a positive correlation with the Nutri-Score are fruit, vegetables, and nuts; however, as we stated before, they have over 95% of missing values.

We can now plot the features against each other to see if they have a correlation. To better visualize this, we plot the correlation matrix.

Salt and Sodium have perfect correlation, and so do energy_100g and energy-kcal_100g (logically it's just a difference in unit measure). Some other features have some degree of correlation like fat_100g and saturated_fat_100g (one is a subset of the other).

We now proceed to check categorical features. The categorical features we take into exam are

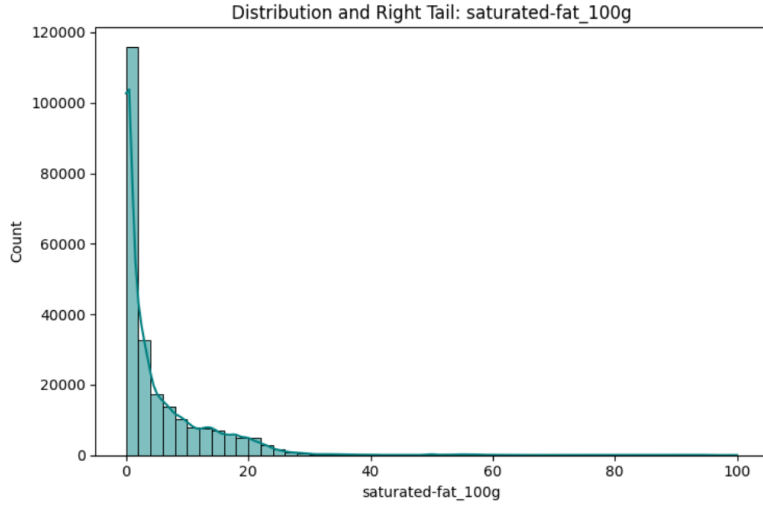


Figure 4: Example of right tail: distribution of a nutrient showing zero-inflation and a long right tail.

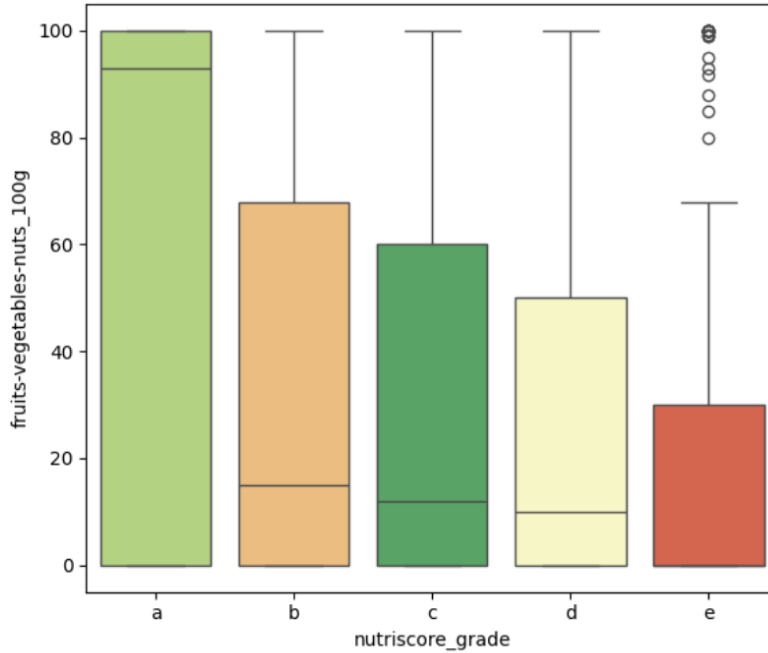


Figure 5: Distribution and importance of the fruits, vegetables, and nuts feature, which suffers from high missingness.

Brands, Product Name, Countries, pnns_group_1, pnns_group_2, categories, and main_category. Of these features, most of them have very high cardinality with a lot of unique values. For our models, we can mainly focus on the ones with acceptable cardinality.

- pnns_groups_1 (10 unique)
- pnns_groups_2 (40 unique)
- countries (3000 unique)

The countries feature has high cardinality and contains acronyms, country names in multiple languages, and inconsistent syntactical formats. Furthermore, many entries consist of multi-label strings where products are associated with several countries simultaneously. To address this, we

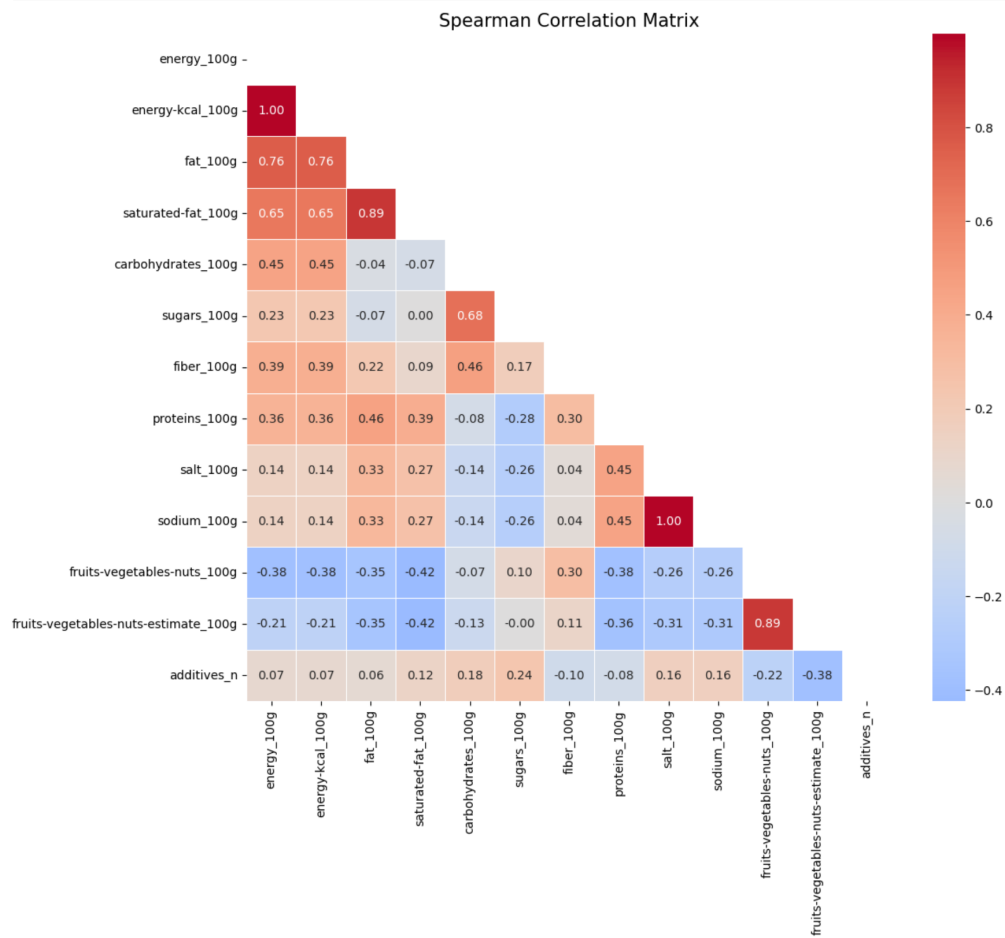


Figure 6: Correlation matrix among the main nutritional features. Highly correlated pairs such as salt/sodium and energy (kcal)/energy (kJ) are evident.

will subsequently implement a targeted approach to standardize these strings and achieve an effective encoding.

Now that we have gained some insights, we can proceed to the preprocessing stage.

4.3 Preprocessing

The preprocessing is done through a scikit-learn transformation pipeline [3]. The pipeline is composed of the following steps:

1. Missing Value Handling
2. Outlier Detection and Removal
3. Categorical Feature Encoding
4. Feature Engineering
5. Feature Scaling
6. Dimensionality Reduction (PCA)
7. Data Splitting

4.3.1 Missing Value Handling

First we drop the features with more than 95% of missing values. In our case, we dropped the features `fruits-vegetables-nuts-estimate_100g` and `fruits-vegetables-nuts_100g`. Remember that these features are important for the calculation of the Nutri-Score with the deterministic formula, so their absence is the exact challenge we are trying to solve with this project. The missing value handling is done through a median imputation. The median imputation is done for the numerical features. For the categorical features, the missing values are labeled as 'unknown'.

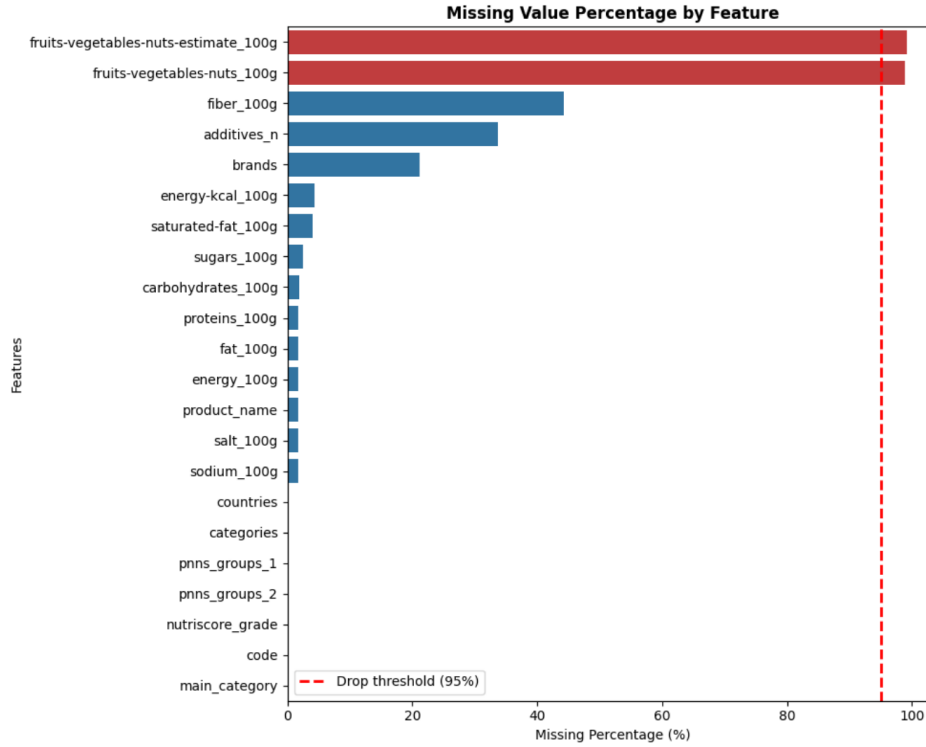


Figure 7: Feature missing values: reasons for dropping 'fruits-vegetables-nuts' and its estimate.

4.3.2 Outlier Detection and Removal

The outlier detection and removal is done through a domain-based and an optional statistical outlier detection. The domain-based outlier detection is done for the numerical features, invalid values such as negative values or values outside the valid range (200g in the column `saturated-fat_100g`). The project also implements a statistical outlier removal method. However this method did not produce good results because the domain of nutritional values is made of highly skewed values with a lot of outliers that should not be removed. For example olive oil is made almost entirely of fats while other foods don't have fat at all, this is not a valid reason to remove olive oil from the dataset. If the statistical outlier is implemented it removes around 25% of data from the samples and models give worse performances overall.

4.3.3 Categorical Feature Encoding

The categorical feature encoding operates on the features `countries`, `pnns_groups_1`, and `pnns_groups_2`. The `pnns_groups_1` feature is encoded using a one-hot encoding, while The `pnns_groups_2` feature is encoded using a target encoding. The target encoding is used because it is a high cardinality feature with a lot of unique values. Target encoding replaces each categorical label with the mean of the target variable for that specific category, effectively converting

discrete strings into numerical values based on their statistical relationship with the output. The **countries** feature presents a more complex case. The feature is a multi-label feature, and it has a lot of acronyms and country names in multiple languages. To address this, we implement a mapping of the country names to a canonical form. This is done by using a generated mapping and the PyCountry library [5]. After this, we apply a multi-label binarization and we choose the top 15 countries by frequency (the top 15 represent 98% of the data) and the other is set to 'unknown'.

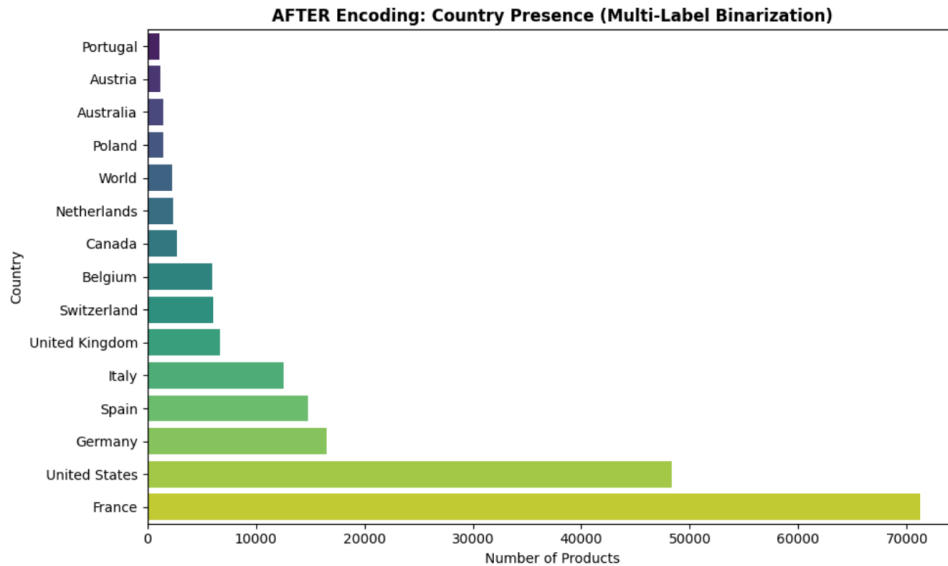


Figure 8: Visualizing the countries feature after cleaning, mapping, and binarization.

4.3.4 Feature Engineering

The feature engineering is done through a transformer that creates the following types of features:

- **Nutritional Ratios:** 4 features:
 - `fat_to_protein_ratio`
 - `sugar_to_carb_ratio`
 - `saturated_to_total_fat_ratio`
 - `energy_density`
- **Energy Decomposition:** 4 features:
 - `calories_from_fat`
 - `calories_from_carbs`
 - `calories_from_protein`
 - `energy_density`
- **Health Threshold Flags:** 3 features:
 - `high_sugar`
 - `high_salt`
 - `high_fat`

4.3.5 Feature Scaling

The feature scaling implements three different scalers: StandardScaler, MinMaxScaler, and RobustScaler. Some features that are highly skewed are transformed with a \log_{1p} transformation before the scaling.

The \log_{1p} transformation is applied to highly skewed features before scaling, which helps reduce the impact of extreme outliers and normalizes the distribution. The mathematical formula for the \log_{1p} transformation is:

$$x' = \log(1 + x) \quad (1)$$

where x is the original feature value and x' is the transformed value. This transformation is numerically stable for small x and prevents issues with zeros in the data while solving the problem of very big data (example the engineered feature `fat_to_protein_ratio` if the product has a small value of protein like 0.01 it will explode).

Through numerous testing of the models, we found that the StandardScaler is the best scaler for the dataset. This is probably because of the nutritional data distributions. Outliers are important so RobustScaler is not the best choice and MinMaxScaler is not the best choice with high skewness.

4.3.6 Dimensionality Reduction

The dimensionality reduction is done through Principal Component Analysis (PCA). The wrapper class first filters all numerical features and sets aside the metadata features (`product name`, `brands`, `code`). The variance threshold is set to 0.95. This means that the PCA will retain the features that explain 95% of the variance in the data. This is done to reduce the dimensionality of the data and to reduce the number of features. The reduction is around 25% of the features (lower variance threshold have been proven to be not good for the model performance).

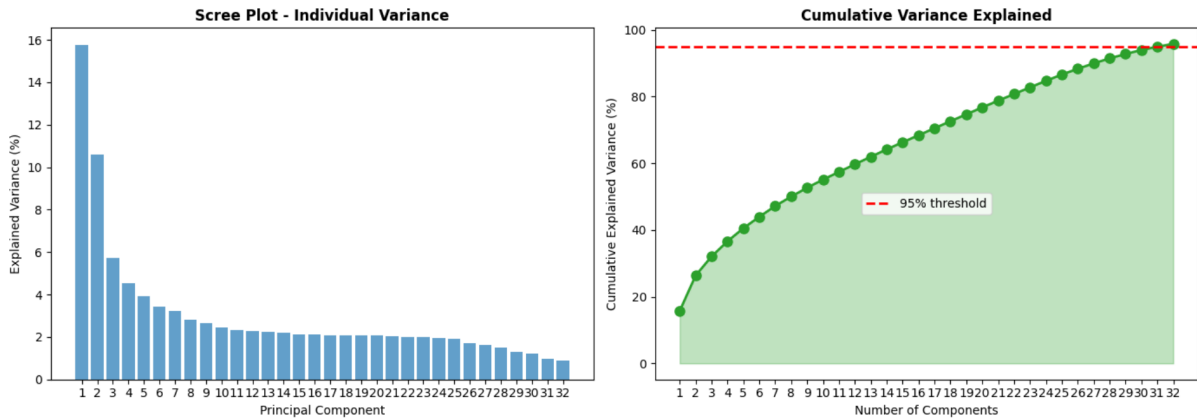


Figure 9: PCA: Explained variance ratio and feature reduction by principal component analysis.

4.3.7 Data Splitting

The data splitting is done through a train, validation, and test split. The split is stratified by the target variable to ensure its distribution is the same in all sets [3]. The split is done using a 70/15/15 ratio. The splits are necessary to evaluate the model performance and to tune the hyperparameters. To prevent data leakage and avoid data snooping bias, we strictly isolate the training set for model development and parameter estimation, ensuring that both the test sets remain completely unseen during the entire development and tuning process.

4.4 Models

In this project we explored the following models:

- **Logistic Regression**
- **K-Nearest Neighbors**
- **Support Vector Machine**
- **Random Forest**
- **XGBoost**

Each model is encapsulated in its own wrapper class, all of which refer to a common BaseModel abstract class interface. The wrapper class is responsible for the training, prediction, and evaluation of the models. This approach allows for easy addition of new models and greatly helped us during the development of the project.

4.4.1 Logistic Regression

The logistic regression is the linear model we use as a baseline. It is a simple model that is easy to understand and to interpret and helped us with the initial testing of the project structure and the preprocessing pipeline. Logistic Regression computes the probability of a product of being in a certain Nutri-Score grade using a linear combination of the features. It is quite simple and does not perform very well in the presence of non-linear relationships.

4.4.2 K-Nearest Neighbors

The K-Nearest Neighbors is a non-parametric model based on the distance. We decided to use the K-Nearest Neighbors to try a distance-based model with our dataset. KNN showed a big gap between the train and the validation accuracy, indicating an overfitting.

4.4.3 Support Vector Machine

The Support Vector Machine is a kernel-based model that is used to find the best hyperplane to separate the data. Since we wanted to explore a non-linear model, we decided to use the Support Vector Machine with a RBF kernel. SVM is a very powerful model that is able to find the best hyperplane to separate the data. Its training time is quite long, but the results are quite good.

4.4.4 Random Forest

The Random Forest is an ensemble learning model. It uses multiple decision trees and the bagging technique to improve the performance and stability. Each decision tree is trained on a random subset of the data and the trees are then used to make a prediction. Studies have shown that Random Forest and gradient boosting models (LightGBM, Catboost, similar to XGBoost) work very well with imbalanced food datasets [4]. As we will see in the results section, Random Forest performed very well with our dataset.

4.4.5 XGBoost

XGBoost (eXtreme Gradient Boosting) is an open-source, distributed machine learning library that implements gradient-boosted decision trees—a supervised boosting algorithm based on gradient descent. It is known for its speed, efficiency, and its ability to scale effectively to large datasets [6].

5 Experiments & Results

5.1 Experimental Setup

We trained all five models on the preprocessed dataset of 250,000 samples with a 70/15/15 train/validation/test split. We used stratified splitting to keep the class distribution consistent across sets, which matters because of the imbalance in our data (class E: 28.2% vs. class B: 11.6%).

We noticed that accuracy wasn't enough to evaluate performance. Because of the imbalance, a model could get decent accuracy just by doing good on the majority classes (D and E) while failing on minority classes. We chose F1-Macro as our main metric because it weights all classes equally.

We started with default hyperparameters and then tuned based on validation results.

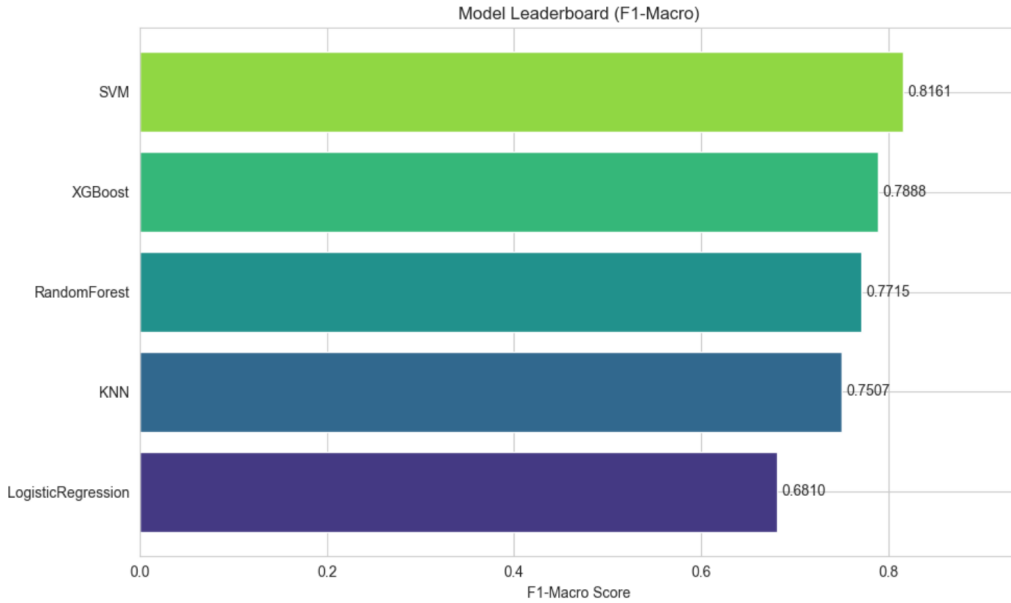


Figure 10: F1-Macro scores of all models on the validation set.

5.2 Performance Metrics

Table 1 shows the results for each model on the validation set.

Model	Accuracy	F1-Macro	Precision	Recall	Time (s)
Logistic Regression	0.702	0.681	0.679	0.685	9.3
KNN	0.771	0.751	0.755	0.749	12.8
SVM (RBF)	0.835	0.816	0.813	0.821	6588.2
Random Forest	0.791	0.772	0.772	0.772	129.7
XGBoost	0.812	0.789	0.797	0.786	36.9

Table 1: Validation set performance metrics and training time for all models.

SVM with RBF kernel got the best results: 83.5% accuracy and 81.6% F1-Macro. Precision and recall are balanced (81.3% and 82.1%), showing no bias toward false positives or negatives. But training took 6588 seconds (about 110 minutes), which was a problem during experiments.

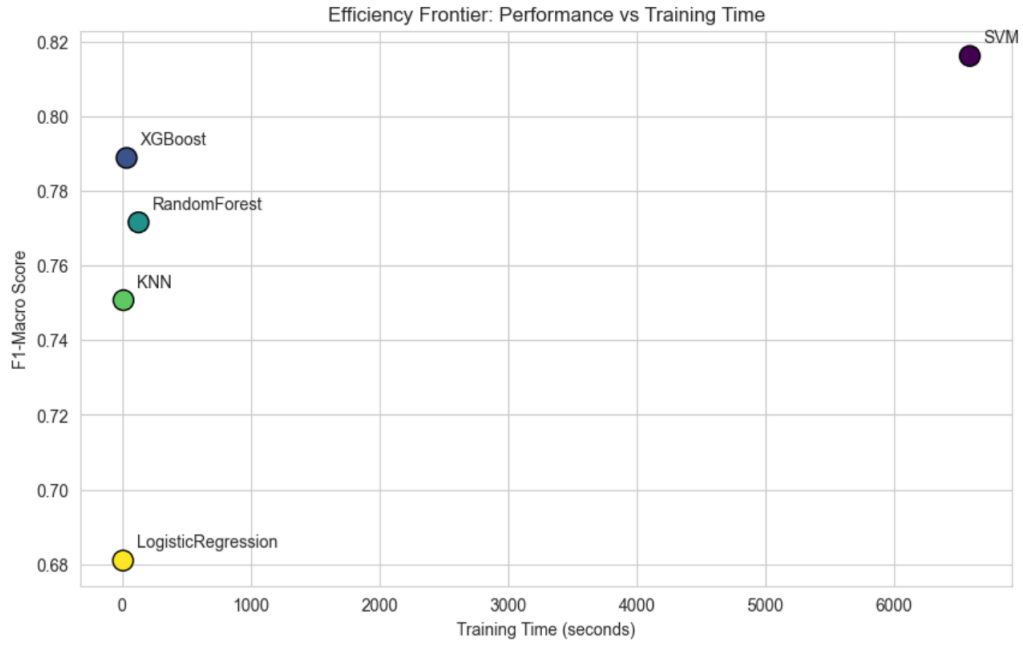


Figure 11: Training time (seconds) for each model.

5.3 Model-Specific Analysis

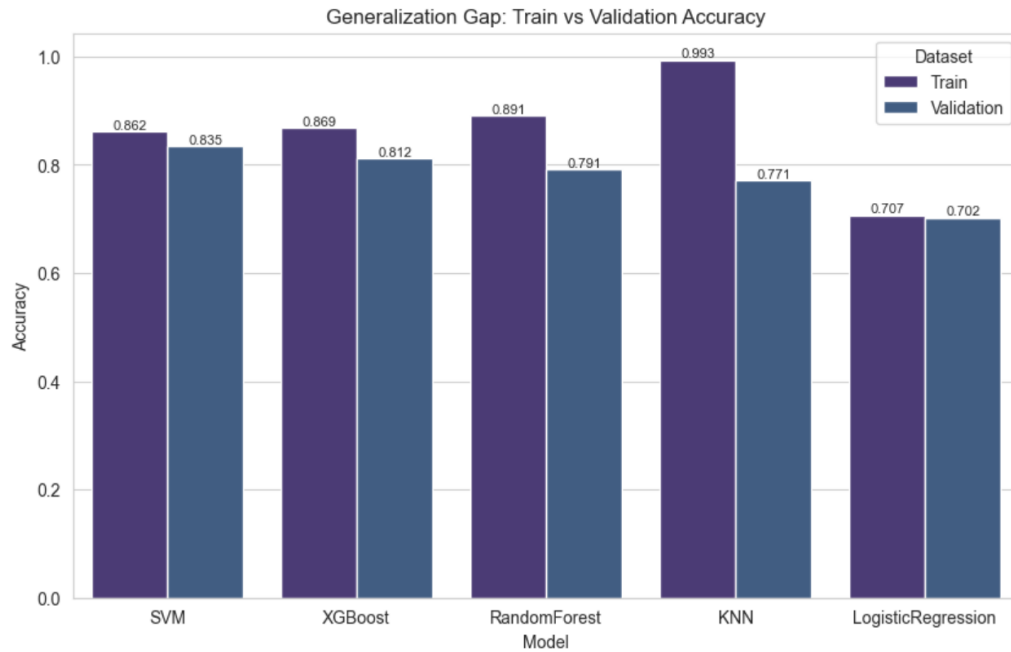


Figure 12: Train versus validation accuracy per model, showing the generalization gap (e.g., KNN overfitting).

Logistic Regression. We started with Logistic Regression as a baseline. It got 68.1% F1-Macro, which confirmed that the relationship between features and Nutri-Score is non-linear (see EDA). The model had difficulty with middle classes (B and C), often confusing them with the ones close to them. This makes sense because the Nutri-Score uses step functions and thresholds, not smooth transitions. We tried polynomial features to add non-linearity, but it only improved F1-Macro to 0.69 while making the feature space much larger.

K-Nearest Neighbors. For KNN, validation F1-Macro was good at 75.1%, but training accuracy got to 99.3% versus only 77.1% on validation – a 22 gap. We tested k values from 3 to 50 and selected k=15, but overfitting remained a problem. The problem seems to be how nutritional data works. Products can look very similar on most features but differ a lot on one nutrient. Since KNN treats all features equally in distance, it just memorizes training examples instead of learning what actually is needed.

Support Vector Machine. SVM with RBF kernel had the best performance at 81.6% F1-Macro. It was very good at handling class B (the smallest class), where the others failed. The RBF kernel works well because it can model complex decision boundaries. The Nutri-Score formula has different thresholds for different nutrients plus interaction effects (like fruits/vegetables / sugar), and RBF captures these. However, 6588 seconds (1.83 hours) training time was too long. We had to limit hyperparameter search because of this. We tried LinearSVC as a faster option, but it got only 0.66 F1-Macro, worse than Logistic Regression. For a real system that needs online updating on new Open Food Facts data, this training time isn’t practical.

Random Forest. Random Forest got 77.2% F1-Macro in about 2 minutes. Random Forest handled class imbalance well. The main issue was that it preferred middle classes (C and D) over extremes (A and E). This happens because when trees disagree, averaging pulls predictions toward the center.

XGBoost. For practical use, XGBoost was the best. It got 78.9% F1-Macro in 37 seconds – the best performance-to-speed ratio. We tuned hyperparameters and found learning rate was critical. 0.3 caused overfitting, 0.01 underfit even with 500 rounds. Best results came from learning rate 0.1. XGBoost also handled missing values well during prediction.

5.4 Cross-Validation Results

The 5-fold cross-validation gave this: (Table 2). Standard deviations were all low (all under 0.003), this means that performance stayed stable across different data splits. We expected more variance because of the noisy data, but stratified splitting and using large folds (about 35,000 samples each) went well.

Model	Mean CV Score	Std Dev
Logistic Regression	0.706	0.001
KNN	0.765	0.001
SVM (RBF)	0.832	0.001
Random Forest	0.791	0.002
XGBoost	0.811	0.003

Table 2: 5-fold cross-validation F1-Macro scores on the training set.

5.5 Error Analysis

Looking at confusion matrices, all models showed similar error patterns. They often confused close grades (C as D, B as C). Big errors (predicting E for an A product) were rare but more common with Logistic Regression and KNN. Class B was the harder to predict – it’s both the smallest class (11.6%) and has a large nutritional range between A and C.

5.6 Observations

Non-linearity matters. The gap between Logistic Regression (68.1%) and non-linear models is a lot. Nutri-Score uses thresholds and interactions that linear models can’t handle well.

Ensemble methods balance performance and speed. SVM performed the best, but Random Forest and XGBoost were very close with much lower cost. The 500x speed difference between XGBoost (37s) and SVM (6588s) is important, thinking of real-deployment use cases.

Feature engineering is important. Our project showed that engineered features added a lot of importance. Nutritional ratios provide context – 10g sugar means different things if total carbs are 80g rather than 15g.

Missing fruit/vegetable data limits performance. This feature is important, especially for distinguishing A from B. We had to drop it (95%+ missing), which probably caused errors in healthy products grades.

6 Conclusions

This project predicts Nutri-Score grades for products in Open Food Facts. We tested five models and found that SVM performed best (81.6% F1-Macro), but XGBoost offered the most practical solution with 78.9% F1-Macro in just 37 seconds.

The results confirmed that non-linear models are best for this project. Logistic Regression only reached 68.1% F1-Macro because Nutri-Score uses thresholds and interactions, not linear relationships. Feature engineering was important – ratio features gave models information they couldn't get from raw nutrient values as they were originally.

The main limitation is missing fruits/vegetables/nuts data (95%+ missing). This feature is important for separating grade A from B, and without it, models had problems with these classes. Class B was the hardest to predict – it's the smallest class (11.6%). Our 250k sample was also a limitation; smaller than the full 1M+ products available from the dataset, but hardware gave us a limit.

For future work, advanced methods could handle missing nutrients better than just dropping them. Testing on the full dataset would show if performance is the same even if the scale is larger. Category-specific models (beverages vs. dairy) might perform better than one general model.

7 References

References

- [1] NCD Alliance. *Unhealthy Diets*. <https://ncdalliance.org/explore-ncds/risk-factors/unhealthy-diets>
- [2] ResearchGate. *Evaluating missing value imputation methods for food composition databases*. https://www.researchgate.net/publication/341157370_Evaluating_missing_value_imputation_methods_for_food_composition_databases
- [3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 3rd Edition. O'Reilly Media, 2022.
- [4] Nalin Arora, Aviral Chauhan, Siddhant Rana, Mahansh Aditya, Sumit Bhagat, Aditya Kumar, Akash Kumar, Akanksh Semar, Ayush Vikram Singh, Ganesh Bagler. *Application of machine learning to predict food processing level using Open Food Facts*, 2025. <https://arxiv.org/abs/2512.17169>
- [5] Flying Circus. *pycountry: ISO country, subdivision, language, currency and script definitions and translations* (Python package version 22.3.5), 2023. <https://github.com/flyingcircusio/pycountry>
- [6] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16), 2016, pp. 785–794. ACM, New York, NY, USA. <http://doi.acm.org/10.1145/2939672.2939785>