CentraleSupélec

DECISION MODELS FOR THE NUTRI-SCORE LABEL OF FOODS

Decision Modeling course project report

Ledia Isaj, Tamara Bojanic, Minato Ben Ahmed Daho

20 November 2020

Contents

1 Introduction

The calculation of the Nutri-Score of food can be considered as a Multi-Criteria Decision Analysis problem. Foods will be considered as the set of alternatives and some aspects of them will be considered as criteria to be maximized or minimized. Energy (KJ), sugars (g), saturated fatty acids (g), and salt (g) are criteria that need to be minimized. Whereas, proteins (g) and fiber (g) need to be maximized.

The goal of this project is to apply different desicion models for this problem and gain deeper insights about them to see which can be a better fit for our problem. In the first part we are going to present and Nutri-Score Model as an additive model based on UTA principles. In the second part we will elaborate a Nutri-Score model based on a simple sorting (ordered classification) model. In the third part we will examine different classification models for this problems. We will explore Decision Trees, Random Forest, K-Nearest Neighbours and XGBoost algorithms.

The code of this project can be found here:
https://github.com/LediaIsaj/Decision_Modeling

2 Nutri-Score model based on an additive model

In order to build an additive model for Nutri-score, we were based on the UTA (UTilités Additives) method proposed by Jacquet-Lagrèze and Siskos (1982). The goal of the UTA approach is to infer additive value functions from a given reference set by using special linear programming techniques.

2.1 Model explenation

The model gets as input an excel file containing data about different products; name of the product, Nutri-score grade, energy, saturated fat sugar, fiber, protein, and sodium it contains per 100 g. We used the PuLP library, which is a Linear Programming modeler in Python. Due to the fact that the input is large, and the performance of the model regarding time decreases exponentially with the input data, if the number of input tuples is bigger than 1000, we randomly sample 1000 products from the original dataset. The sample will be used to build the additive model, and then we will use the model to predict the Nutri score labels for the rest of the data. Linear programming variables were created for the model: products variables, epsilon variables (the difference between two constructive classes), and marginal utility variables. The goal of the model is to maximize the differences between classes.

$$max \sum_{i=1}^{4} epsilon_i$$

Also, the model has constraints regarding the global utility for each food and monotonicity constraints. Also, a preference setting was created, by randomly selecting 5% of the data for each class. This set was used to give a partial preorder using the real Nutri-Score label. These preferences are presented as constraints in our model.

2.2 Experiment and results

The input dataset contains 5604 different products. The model takes a sample from the original dataset to train the model. We experimented to run with different sample sizes. We observed that as the input data grows, the time to solve the LP model increases exponentially. To run with the original dataset would take more than 1 day. For this reason, we randomly sampled 1000 products to train the model. We randomly sampled from the dataset and run the model until we got a solution that was feasible with good precision values. The trained model was used to predict the Nutri scores for the unlabeled data. The results of the model are saved in a CSV file. For scoring new data, we assign the utility constraint closest to the new utility value. In the same manner, we proceed with the final Nutri-grade. The results are saved in a CSV file called label-prediction.

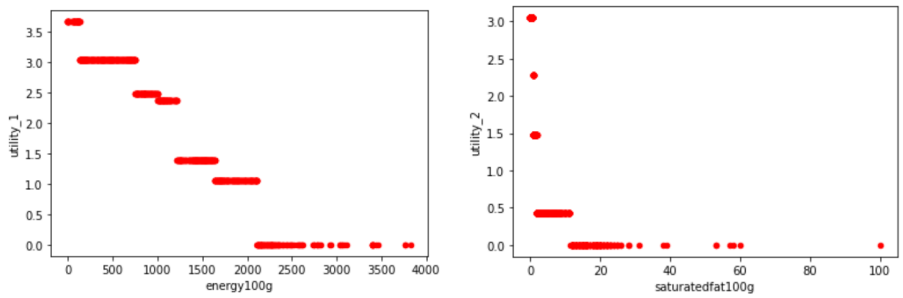

Figure 1: Utility 1 (energy), utility 2 (saturated fat)

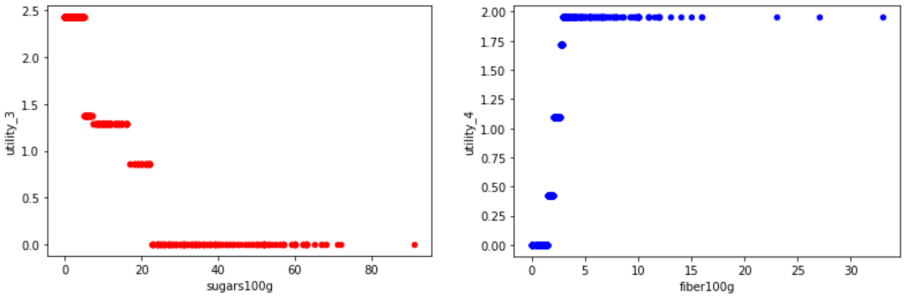

Figure 2: Utility 3 (sugar), utility 4 (fiber)

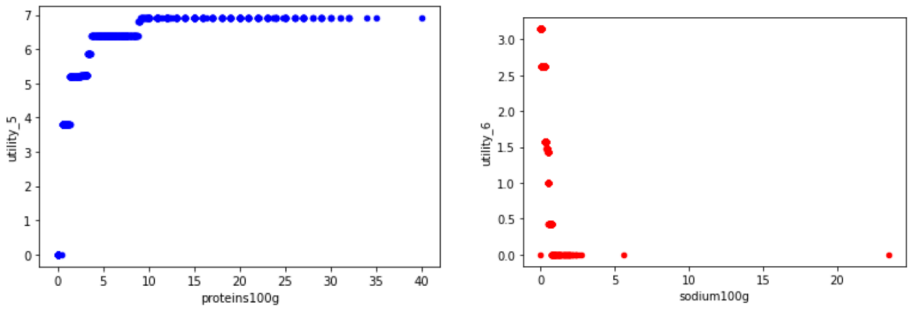

Figure 3: Utility 3 (protein), utility 4 (sodium)

In figures 1,2,3, we can see the utility functions. Energy, saturated fat, sugar, and sodium need to be minimized. Fiber and proteins need to be maximized.

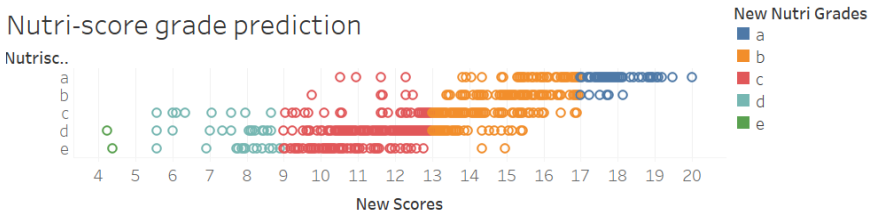

Figure 4: Results of model predicting nutri score

In figure 4 we can see the results of our model. On X-axis, the score predicted by our model, on Y-axis is the real Nutri score, and detailed with the color we can see the Nutri score calculated by our model. As we can see our model tends to be optimistic. However,there are no extremely contradictory results. In the table below, we can see the presision, recall and F-measure for each class.

Class	A	B	C	D	E
Precision	0.6	0.9	0.2	0.7	0.05
Recall	0.95	0.3	0.11	0.42	0.5
F1	0.73	0.45	0.14	0.12	0.1

2.3 Discussion for the additive model

In the section above, we showed one of the experiments. We run the additive model with different size of sampled data and fraction of preference set. When the number of products to train the model increases, the time to solve the model increases almost exponentially. This is one of the main drawbacks of this model. If we have a big dataset, the reference set still needs to be small, otherwise, the time to solve the problem is huge. However, the trained data might be small and not representative of the whole dataset. Secondly, the model requires no-contradiction for constraints in order to run. Nutri-score from different classes had some contradictions. For this reason, we needed to resample from the original dataset until we found an optimal solution.

3 Nutri-Score model based on a simple sorting (ordered classification) model

Another model that was used to predict the Nutri-score was the model based on a simple sorting. The goal of this experiment was to develop two different methods: *PessimisticMajoritySorting* and *OptimisticMajoritySorting*, respectively based on the Pessimistic and Optimistic version of MR-sort rule.

The database of food items that we will be working with to develop simple sorting models contains six different categories: energy, sugar, saturated fat, salt, protein and fiber, as said before.

3.1 Models explanation

Both pessimistic and optimistic model have the same input and the output. The input of the models include the input file with the food items, output file where the food items and its predicted and true nutri-scores are written, weights of the food categories, table of the limiting profiles and the flag that specifies whether or not we should output the results. The weights of the food categories are one parameter of the model, their values range from 1 to 2 and they specify how much influence a category has on the predicted nutri-score. In our case, the total number of "voters" (or the sum of all weights) is always 8. When assigning the nutri-score category to a food item, weights of all food categories where the food item is performing better than a limiting profile are added together, and if that sum is greater than the multiplication of some threshold and the total number of voters (in our case 8), the food item is assigned with the appropriate nutri-score category. The table of the limiting profiles is another parameter of the models and it specifies the range of the nutri-score categories for each food category. The two models differ in the following way: The pessimistic model will assign a nutri-score category to the food item when the item's nutritive value is at least as good as the lower limiting profile of this category and is not at least as good as its upper limiting profile. The optimistic model will assign a nutri-score category to the food item when the upper limiting profile of this category is better than the nutritive value of the food item and the lower limiting profile of this category is not better than the nutritive value of the food item for this category.

3.2 Experiments

Here we describe the implementations of the methods and their improvement. All experiments were ran using three majority thresholds: $\lambda = 0.5$, $\lambda = 0.6$ and $\lambda = 0.7$. The two experiments are tested on the given data set of food products.

3.2.1 Experiment 1: Apply the given limiting profiles and weights

In the first experiment the task was to use the given limiting profiles and weights and apply it to our two algorithms, for pessimistic majority sorting and optimistic majority sorting. The given parameters are shown in figures below.

	1- Energy	2-Sugar	3-Satu. fat.	4- Salt	5- Protein	6- Fiber
π^6 : Upper limiting profile	100	0	0	0	100	100
π^5 : Limiting profile between A and B	1550	11	0.8	0.3	10	11
π^4 : Limiting profile between B and C	1650	14	1	0.4	7	8
π^3 : Limiting profile between C and D	1750	17	1.7	0.5	4	5
π^2 : Limiting profile between D and E	1850	20	4	0.6	3	2.5
π^1 : Lower limiting profile	10000	100	100	100	0	0

	1- Energy	2-Sugar	3-Satu. fat.	4- Salt	5- Protein	6- Fiber
Weights w_i	1	1	1	1	2	2

Figure 5: Given limiting profiles and weights

The code of the PessimisticMajoritySorting and OptimisticMajority sorting are very similar and are available in our github repository.

After we apply the two methods using the given weights, table and thresholds, we get the following results, as shown in confusion matrices below.

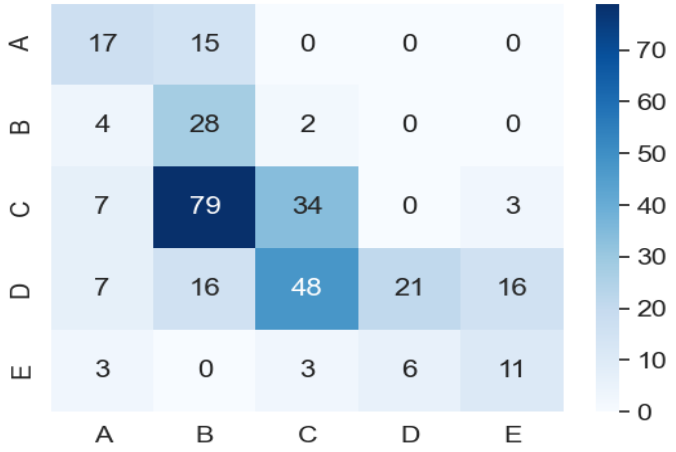

Figure 6: Results for the Pessimistic Model, $\lambda = 0.5$: Accuracy = 0.68

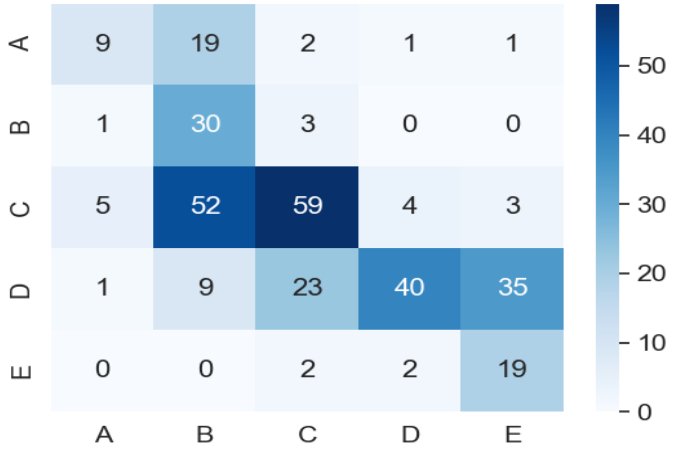

Figure 7: Results for the Pessimistic Model: $\lambda = 0.6$: Accuracy $= 0.72$

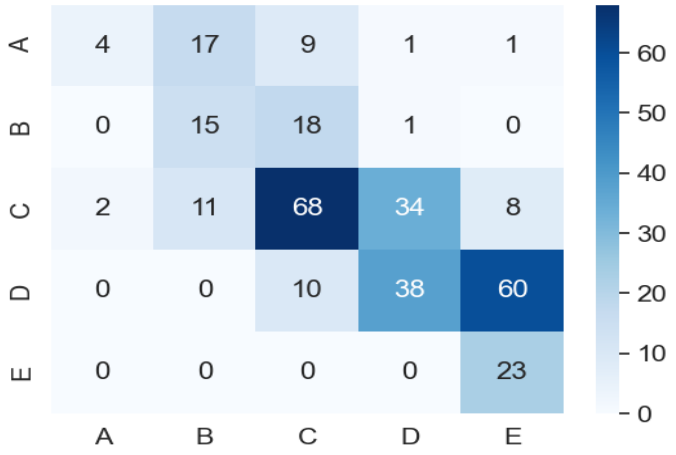

Figure 8: Results for the Pessimistic Model: $\lambda = 0.7$: Accuracy $= 0$.

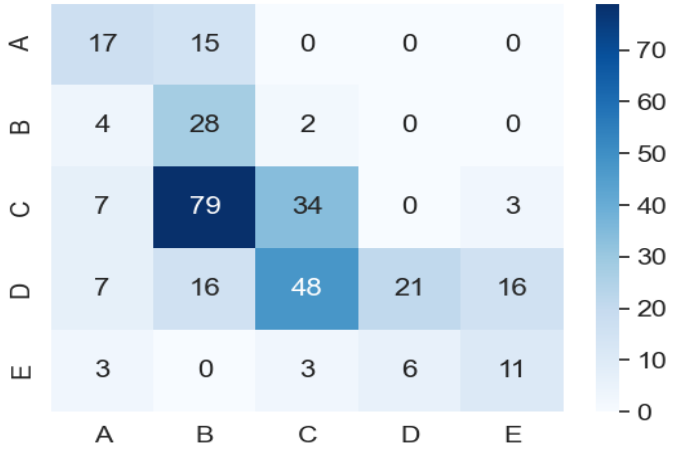

Figure 9: Results for the Optimistic Model: $\lambda = 0.5$: Accuracy = 0.35

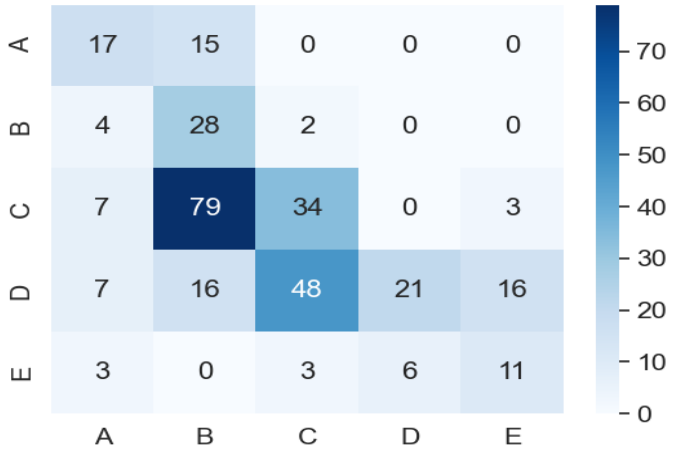

Figure 10: Results for the Optimistic Model: $\lambda = 0.6$: Accuracy = 0.35

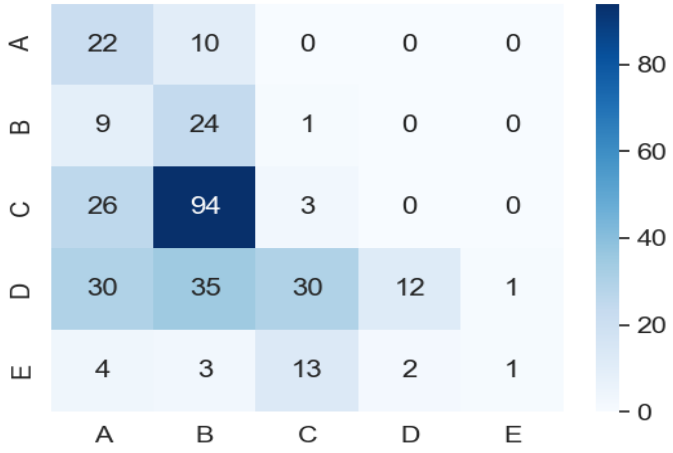

Figure 11: Results for the Optimistic Model: $\lambda = 0.7$: Accuracy = 0.19

As the results are not great, in the next experiment we will try to change the values of the limiting profiles and the combination of weights in order to better classify the food items.

3.2.2 Experiment 2: Improve the given limiting profiles and weights

As mentioned above, in this experiment we try to improve the table of limiting profiles and the value for weights, in order to better classify the food items. More specifically, for each threshold, we will come up with the combination of limiting profiles and weights that classify the data from the given data set with the highest accuracy possible.

The idea is not too complex. We will only explain the process for the Pessimistic Majority Sorting, since the process for the Optimistic Majority Sorting is almost the same. The code is in the file called ordered-classification.py.

First thing important to mention, is that we decided to have the total number of voters equal to 8, so in all cases the total sum of weights would be 8, like in the given example. Also, all weights can be either 1 or 2, for simplicity. So, since we have 6 criteria for food, that leaves us with 15 possible combinations of weights among the food criteria. Then, the next thing we want to do it come up with a smaller set of weights, since training the model on all these sets of weights would take a very long time. To do this, we simply run the pessimisticMajoritySorting with all sets of weights and keep only the 5 combinations that give the highest accuracy. Then for each of the 5 weights combinations, we run the improved version of the pessimistic majority sorting algorithm.

This version of the algorithm starts with the given table of limiting profiles

and tries to change its values to better classify the data. We first need to copy the current table of limiting profiles to some temporary table. For each cell of the table (except for the first and last row) do the following:

- take the difference between the cell and the one **above** it, divide it by 2 and subtract from previous value - if the values in that column are increasing (first 4 columns), or add to the previous value - if the values in that column are decreasing (last 2 columns)
- update the cell with the correct value
- run the pessimisticMajoritySorting and obtain the accuracy

as shown in the code below:

```
if col < 4:
    new_table1[row][col] -= round((new_table1[row][col] -
        new_table1[row - 1][col]) / div, 2)
else:
    new_table1[row][col] += round((new_table1[row - 1][col] -
        new_table1[row][col]) / div, 2)
pairs1 = pessimisticMajoritySorting(inputFile,
    "pessimistic_OpenFood_Petales.xlsx", weights, new_table1,
    threshold, False)
accuracy1 = calculateAccuracy(pairs1)
```

- then take the difference between the cell and the one **below** it, divide it by 2 add to the previous value - if the values in that column are increasing (first 4 columns), or subtract from the previous value - if the values in that column are decreasing (last 2 columns)
- update the cell with the correct value
- run the pessimisticMajoritySorting and obtain the accuracy

```
new_table2 = copy.deepcopy(table)
if col < 4:
    new_table2[row][col] += round((new_table2[row + 1][col] -
        new_table2[row][col]) / div, 2)
else:
```

```
        new_table2[row][col] -= round((new_table2[row][col] -
            new_table2[row + 1][col]) / div, 2)
    pairs2 = pessimisticMajoritySorting(inputFile,
        "pessimistic_OpenFood_Petales.xlsx", weights, new_table2,
        threshold, False)
    accuracy2 = calculateAccuracy(pairs2)
```

Then choose the best accuracy between the one before changes and the two that we obtained as shown above and if one of the new accuracies is the best, update the so called "best table" that will be used in the next iteration. We run the improved version of the method until convergence, or as long as the limiting profiles are changing.

In the end, the sets of weights and limiting profiles that gave the best results for each threshold are the ones shown below:

	1- Energy	2-Sugar	3-Satu. fat.	4- Salt	5- Protein	6- Fiber
Weights w_i	1	1	1	**2**	**1**	2

Figure 12: Produced weights for the improved version of Pessimistic Majority Sorting

1-Energy	2-Sugar	3-Satu. Fat	4-Salt	5-Protein	6-Fiber
100	0	0	0	100	100
1550	5.5	0.8	0.3	8.5	55.5
1556.25	15.5	0.85	0.37	8.5	31.75
1675	18.5	1.7	0.43	6.25	5
1762.5	20	4	50.3	4.62	2.5
10000	100	100	100	0	0

Figure 13: Produced limiting profiles for the improved version of Pessimistic Majority Sorting

The results for our improved version of the pessimistic model are shown below:

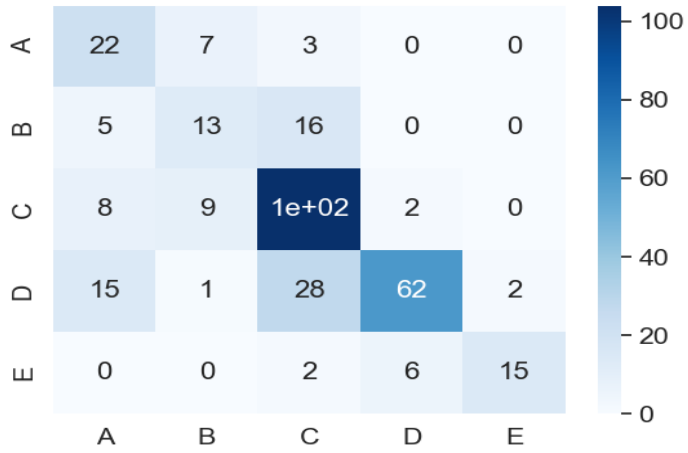

Figure 14: Results for the improved pessimistic model: $\lambda = 0.5$: Accuracy = 0.68

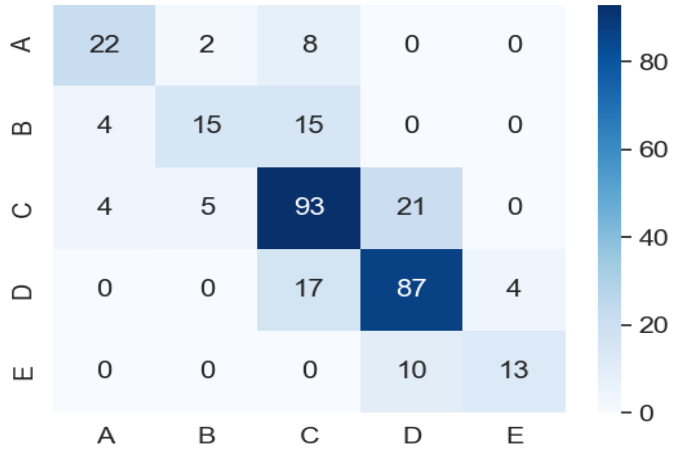

Figure 15: Results for the improved pessimistic model: $\lambda = 0.6$: Accuracy = 0.72

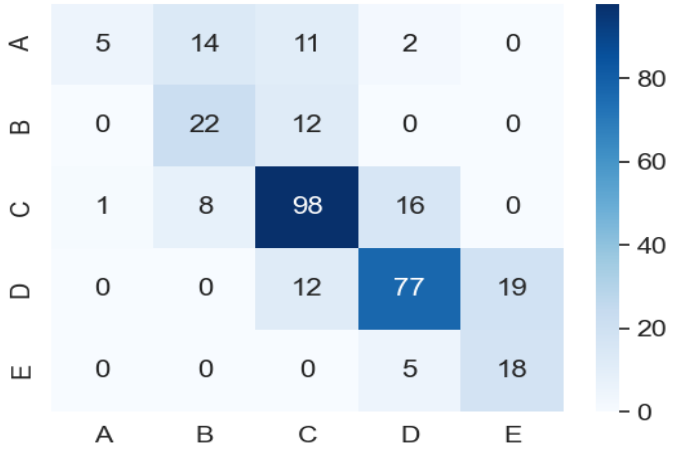

Figure 16: Results for the improved pessimistic model: $\lambda = 0.7$: Accuracy = 0.69

4 Nutri-Score model based on machine learning model

The idea of this section is to build several machine learning base models and to choose the most effective one, effectiveness will be defined. Therefore, after the exploratory data analysis, we will test several models based on the data shape and then we will build the final required function.

4.1 Exploratory Data Analysis

4.1.1 Cleaning the dataset

We will use the openfoodfacts database for choosing our model and then we will implement it on the cereales dataset. After loading the data set, we dropped all non-useful columns and rows with NA. Then, as ML models supports only numerical values, we replace nutri-score classes letters by numbers (from A-E to 0-4), A=0 and E=4.
A machine learning model works with several features and a label. Here, the label y is 'nutrition-grade-fr' and the features X are 'energy-100g', 'saturated-fat-100g', 'sugars-100g', 'fiber-100g', 'proteins-100g', 'sodium-100g'. Then, each set must be split into a training set, a validation set and a testing set. Therefore, at the end of the day we have:

- Train set: 3395 values (85% of the total dataset)
- Validation set: 360 values (9% of the total dataset)
- Test set: 240 values (6% of the total dataset)

4.1.2 Exploratory Data Analysis

First, we must see how the data is distributed to see if there are some outliers or extreme values. Here are some boxplots.

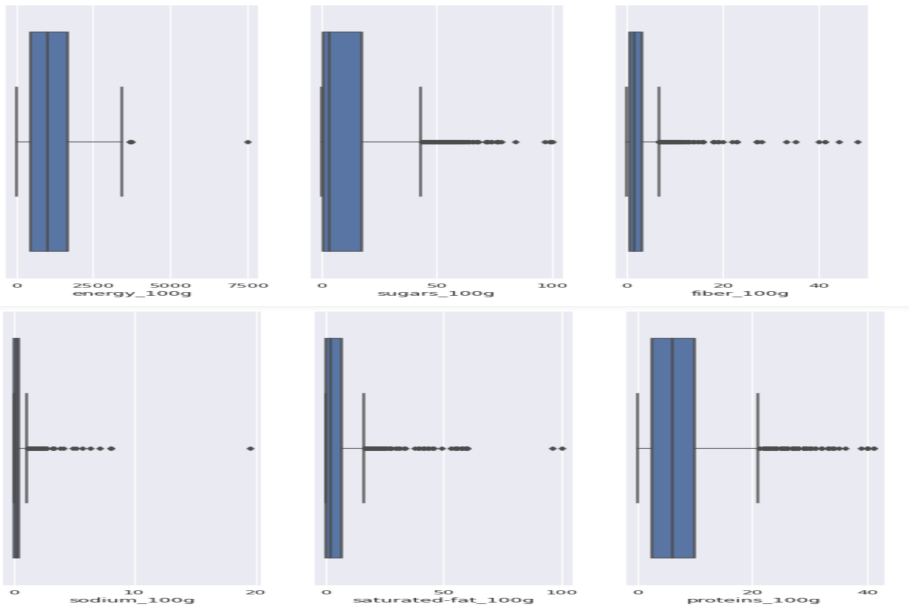

Figure 17: Boxplots of food features

As we may see, energy is the feature with the less extreme values contrary to sugars, fiber and proteins. Moreover, energy and proteins seem to have more dispersed values than sodium and fiber. These conclusions may give us some insight on what features are more determinant when classifying our foods. Flatten boxplots mean that values are quite similar. On the other hand, a large boxplot means that the dispersion is higher. Therefore, we may think that energy, sugars and proteins would be the first discriminating criteria.

Finally, in each set we can see that each feature doesn't have many aberrant points. Outliers are not that numerous and may be explained because the list of foods contains some primary ingredients like sugar or water. Therefore, it could be useful to keep these values so have a standard of what pure ingredients means that what food grade is associated with these extreme values.

Next, let's quickly have a look on correlations between features and label. We plot the correlation heatmap.

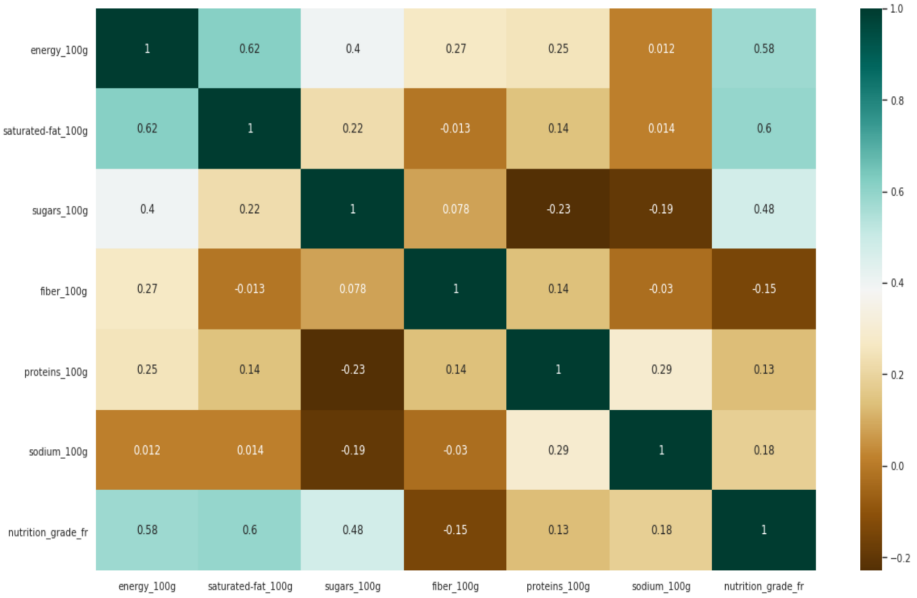

Figure 18: Heatmaps of features and label

Although we are going to use classifying models, this matrix could be very useful to understand how the current model work and how our models will try to copy it. Here we can see that energy, fat and sugar are the greatest drivers of nutrition grade - the higher they are, the higher the grade will be, which means it will rise to 4 (E), lowest grade. Moreover, we can see that energy is quite correlated with fat and sugars, but fat and sugars are not. This gives us more insight on how a grade is picked but also how the food composition works. It is obvious that the more sugars or fat you had the more energy you will have, but something sweet is not necessarily greasy. Therefore, adding all insight we may think that energy and sugars are the principles drivers, but we will find out when implementing models. In the code we added some plots just to have a better understanding of distribution and correlations by plotting each variable against the others. For example, we can see that classes are not that equally distributed. There is way more class A and D products then B and E.

4.2 Decision Trees

In the previous section, we saw the shape of our dataset, less than 100k values. Therefore, the first model that we may think of is Decision Trees.

4.2.1 Model explanation

Our model does not need any hyperparameter adjusting, so we fit it directly to our train sets. On the validation set, it has an 79.2% score of accuracy. To avoid any overfitting issues, we use the cross-validation method with 5 subsets, which also gives us an average score of 78.9%. We will also be able to compare our models.

To understand how the model works and what are the main nodes, we printed the tree using different methods (mathplotlib, graphviz and an online dot to graphviz converters). As we may see, the tree is long, and some branches are way too long to be readable. Therefore, we will focus only on the first branches (full tree is available as a svg file to have a better resolution).

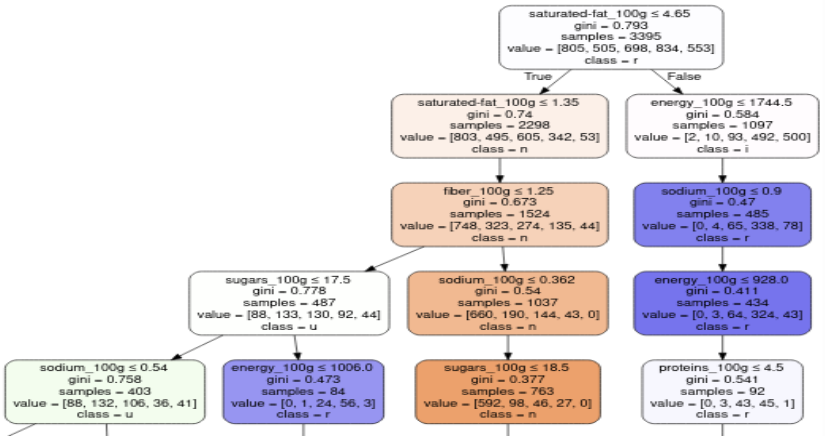

Figure 19: Partial Decision Tree

Here we kept only some part of the tree, but it still helps understand the first steps. Fat and energy are the first three nodes. However, it is quite interesting to see that fiber and sodium comes next. In our exploratory analysis we though that they may not be such discriminating factors, but it turns out that they are used pretty son in the tree, before sugars. It is quite interesting when you confront this tree to a human decision process, because the first thing that comes to a human mind when you talk about nutri-scrore and healthy food is fat. It would be very interesting to compare this tree with how a human being would draw its one. It will probably take fat, then sugar or energy because we want

to eliminate the bad ones straightforward as there are easy to identify. Sodium and fiber don't immediately come to mind but the algorithm found that they were useful to classify food. Let us check the results.

4.2.2 Results

As mentioned before, the overall accuracy of the model was 78.9%. Let us have a look on the normalized confusion matrix. We evaluate now the model on the test set.

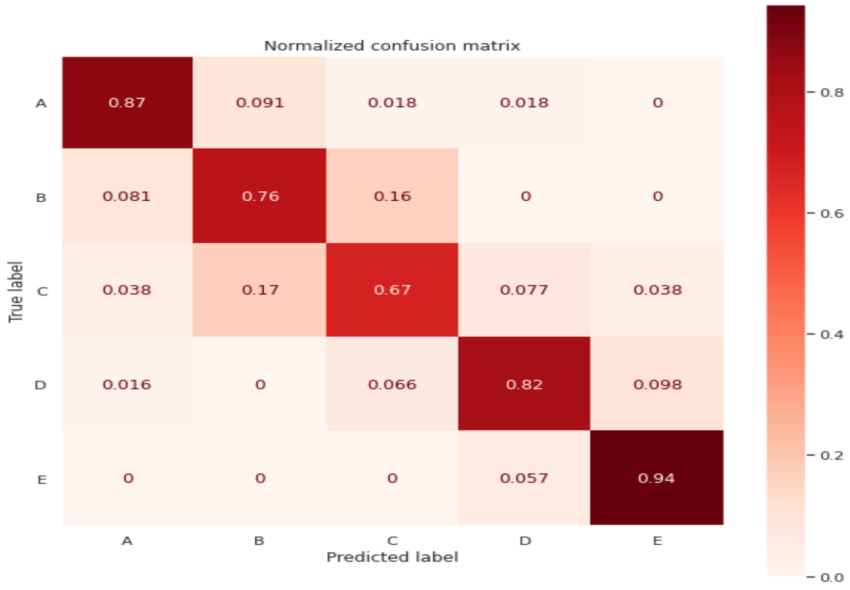

Figure 20: Normalized Confusion Matrix for the DT model

As expected, our model is efficient when classifying worst and best products. However, if take a closer look at the square B-C we can see that the confusion is higher. This means that products labelled B or C are often mistaken for one another. Therefore, we cannot conclude that our model is efficient enough. Although the overall score in encouraging and the accuracy in extreme values is high, the confusion between medium products is too high. Therefore, we can test another model.

4.3 Random Forests

The second model we will try is Random Forests. We will see how to choose the hyperparameter and what are the results.

4.3.1 Model explanation

Before adjusting hyperparameters, let's fit the model using the default value n-estimators=100. We obtain a score of 85.6%. Using the same method as previously the average score is 85.2%. We may say that this score is higher than the previous one, but we used the default value for the number of estimators. Let us plot the validation curve to see if a better value of the hyperparameter exists.

Figure 21: Validation curve for the RF model and the hyperparameter n-estimators

Looking at this curve we see that there aren't any huge increases after n=75 in the overall score. Therefore, we can keep our default value, which was a 100. On average, the score variate from 84 to 86%, which is still higher than our previous model. We could have use gridsearch to find the perfect value but as scores as slightly the same and we are still searching for the best model, we can keep n=100. Let us see in detail the performances of our model.

4.3.2 Results

The overall accuracy of the model was 85.2%, which is better than previously, but does it perform better one the middle classes. Let us analyse the normalized confusion matrix plotted as before using the test set.

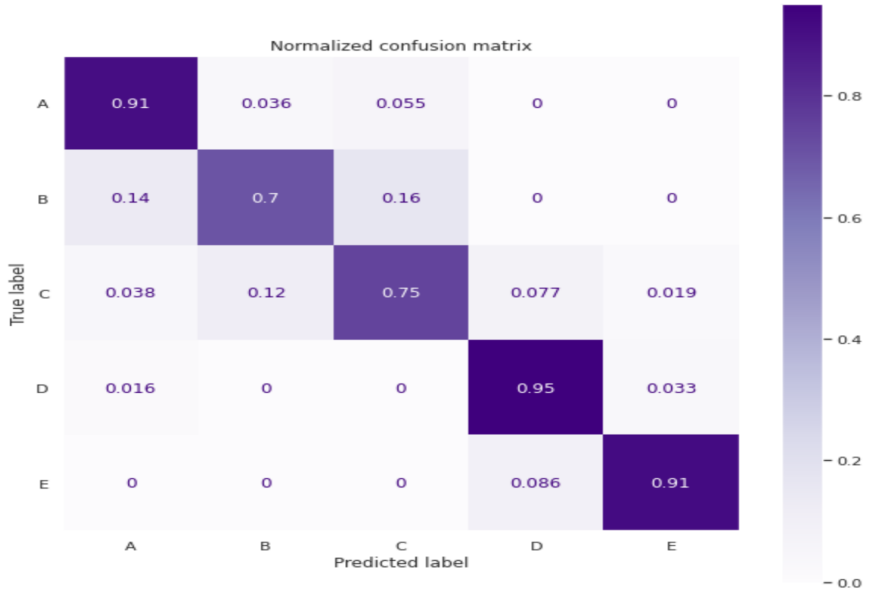

Figure 22: Normalized Confusion Matrix for the RF model

As we may see, we have improved our accuracy for most classes. Huge mistakes are less frequent, we have non null numbers around the diagonal but not at the corners of the matrix. In the previous one we had some A products mistaken as D products for example. However, even though class A, D and E have more than 90% accuracy, the score for class B and C is still lower than expected. It seems like it is difficult for the model to differentiate between them. moreover, a new problem occurred, B products are also mistaken for A products. This error wasn't present in the previous model. But unlike B-C this error is only one way. B and C are mistaken for one another, but an A product is rarely given a predicted label B. This maybe means that adding to the B-C issue, our model tends to overrate products. The same overrating situation can be seen between D and E. Let us try other models before choosing the final one.

4.4 Non-successful models

4.4.1 K-Nearest Neighbours

We tried the KNN model first. To choose hyperparameters, we tried to score the model using the same method. According to our results, we decided to pursue using distance weights. Then, we plotted the validation curve to choose the optimal value for k. However, accuracy scores didn't pass 65%. Therefore, we decided to abandon this model. As our dataset as less than 100k values, it was too small for this model to fit perfectly.

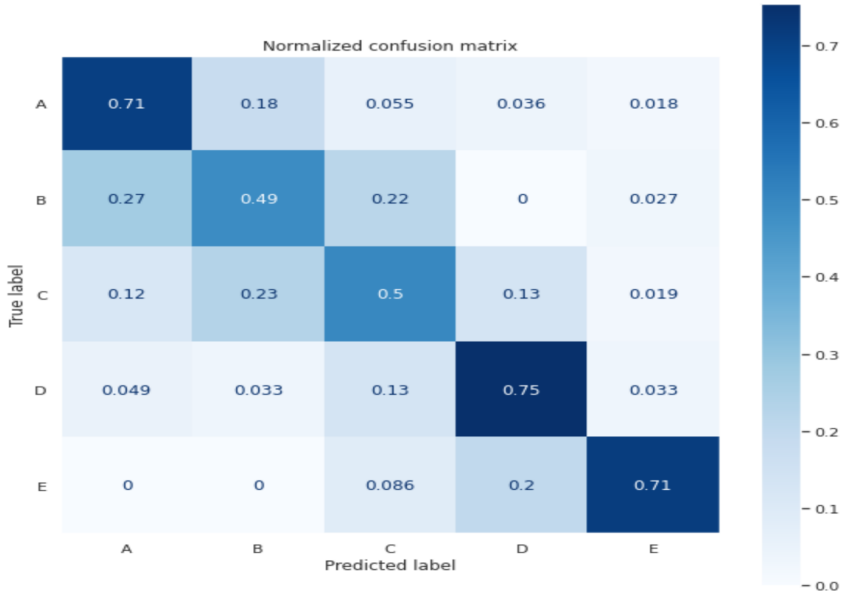

Figure 23: Normalized Confusion Matrix for the KNN model

Moreover, the results were bad. if we look at the normalized confusion matrix, we can see that accuracy scores are low, and many errors appeared in the corners of the matrix, especially top-right. This means that the model tends to drastically underrate products. Therefore, we did not keep this model.

4.4.2 Support Vector Machine

Reasons why the model was not fitting were pretty much the same as before: the dataset is not large enough. Moreover, the score was too low to obtain a good model. Therefore, we did not keep this model either.

4.5 XGBoost

4.5.1 Model explanation

XGBoost is a gradient boosting algorithm that will combine several models to have a better result. The idea is that at each iteration the model will reevaluate products and give them a weight base on their contribution to predictions. First, we took the default values for hyperparameters and we obtained a score of 85.3% accuracy on the validation set. Then we adjusted our models with many hyperparameters such as the depth or the number of classes we wanted. After adjustment, the average score of the model 86%. This model unable us to have an importance score for each feature. Thereby, we can understand it better.

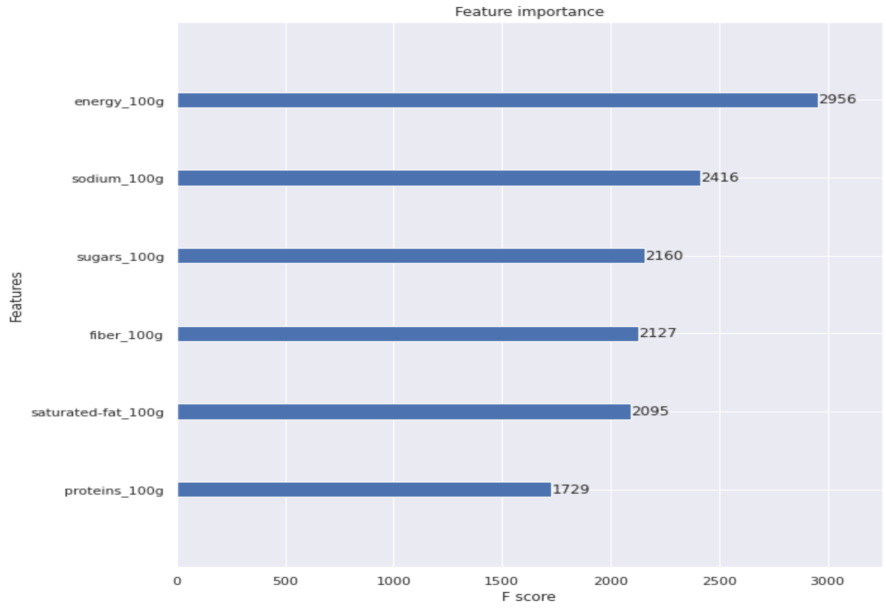

Figure 24: Feature importance for the XGB model

This model, like previous ones, puts a high importance on energy. However, surprisingly, unlike what we though from the distribution, sodium is important. One may think that it is a useful criterion to differentiate two similar products in other features. Sugar comes third as we expected. Then we have fiber which was also unexpected. If we try to understand this in a human point of view, we may think that fat and protein are not that important to classifying because we associate fat with bad and proteins with good. Thereby, if a product has fat it will be bad but how much bad? Well we look at other criteria like energy or

sodium. Same thing goes for proteins. This model could be a good model to reproduce human preferences. But let's look at its results.

4.5.2 Results

The overall accuracy of the model was 86%, which is the higher we got. Let us break this result in more specific details with the normalized confusion matrix.

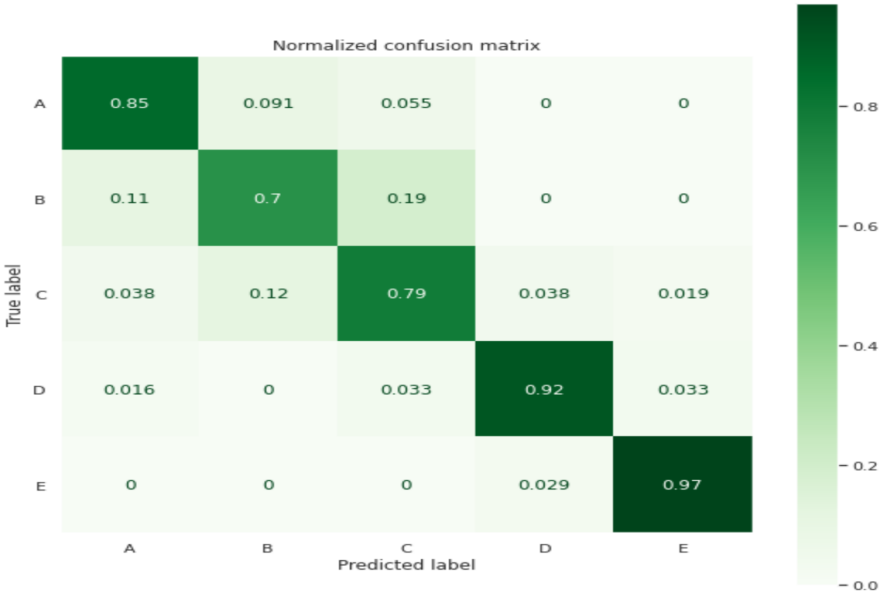

Figure 25: Normalized Confusion Matrix for the XGB model

As for the previous models, this model can identify bad products. However, unlike the RF model it underperforms on good products. However, the score on B and C class products is quite good. Results are overall quite like what we had with RF model. Therefore, we may need another metric to compare them for our final choice. Because although the accuracy is higher, the difference is not large enough to say that this model overcomes the RF one.

4.6 Final function

4.6.1 Final model

Let us compare the two models, RF and XGB, with accuracy, precision and recall. We will take the weighted average on classes for each metric. Precision

is defined as the number of true positives over the number of true positives plus the number of false positives. Recall is defined as the number of true positives over the number of true positives plus the number of false negatives.

Model	Accuracy	Precision	Recall
XGB	85.98	85.07	85.00
RF	85.15	85.16	85.42

Figure 26: Comparison of XGB and RF models

We decided to keep the RF model as our final model because overall the score was better. Moreover, the RF model had better results with B-C confusion.

4.6.2 Final function and tests

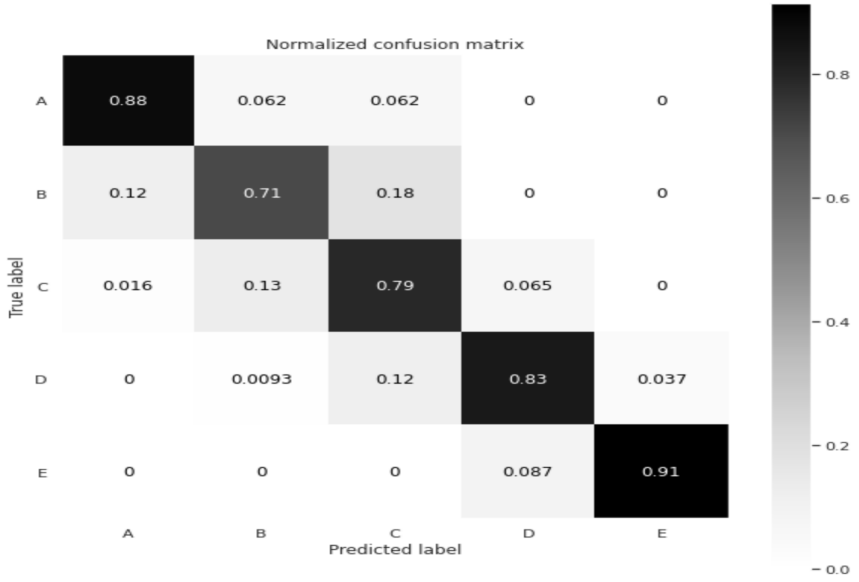

Figure 27: Normalized Confusion Matrix for the RF model and final dataset

We defined the final function and tested it on the Cereales dataset. Finally, the model performed quite well on the dataset, especially on the extreme values.

5 Conclusion

The additive model tends to be optimistic in the results, meaning it gives a bit higher nutri-score grade comparing to the real one. However, there are no extremely contradictory results. One of the main drawbacks of the additive model is that it takes a long amount of time to solve as the number of input data increases. Therefore, we need to sample, and if the sample is very small compared to the real one , the sample might not be representative enough for the whole dataset. Secondly, the model requires no-contradiction for constraints in order to run. Nutri-score from different classes had some contradictions.
The simple sorting model performed much better once we changed the parameters. However, none of the experiments showed better results than 0.72, which tell us that the algorithm always gets stuck at the local optima. One way to solve this is to change the "starting point" of the table of the limiting profiles. In our experiments, we always took the given table of limiting profiles as the starting table that would be modified during iterations, and we have reasons to believe that the algorithm would perform even better if it was initialized with different sets of parameters. However, if we try to change all possible parameters in order to find the best combination, the model becomes very complex. With this approach it would be better to use neural networks or some other existing machine learning algorithm.
The machine learning model tends to perform good on the extreme values. However, on the classification of B and C products the model tends to mix them. Moreover, the model tends to overrate some products. Machine learning models helps us understand the importance of each features. As our first insights showed, energy and sugar are the most important features, but surprisingly sodium is also (in the tree and the plot). To explain the consumer choices, the machine learning model helps us understand which criteria are the most significant. As we said it before, some models understand that fat gives a low rating and therefore, they categorized directly the product as bad then they temper this classification using other features more versatile like sodium or energy. However, these models are not very explainable, especially to a consumer. The DT is too wide and has too many leafs, RF also, KNN is not working and XGB is too complex.