Dauphine | PSL
UNIVERSITÉ PARIS

Rapport de projet

L'algorithme de calcul du Nutri-Score des produits alimentaires

Auteurs :
Shahram MAHDAVI
Charly SIMONIAN

Encadrant :
Brice MAYAG

10 Décembre 2019

Table des matières

I) Introduction :

L'objectif de ce projet consiste à analyser et interpréter la transparence des modèles d'évaluation utilisés pour la classification de produits et plus particulièrement Nutri-score. Ce dernier est un système d'étiquetage nutritionnel pour faciliter le choix d'achat du consommateur en fonction de la composition nutritionnelle des produits.

Avant de pouvoir analyser la transparence des différents algorithmes de Nutri-score, il est nécessaire de savoir ce qui est un Nutri-score et comment cela marche.

Par la suite, nous allons nous concentrer sur les différents algorithmes de calcul et puis nous allons discuter sur l'aspect transparence de ces algorithmes au point de vu des consommateurs qui ne sont pas forcément des informaticiens.

Le Nutri-score est un logo qui informe sur la qualité nutritionnelle simplifiée et complémentaire à la déclaration nutritionnelle obligatoire fixé par la réglementation européenne. Le nutri-score est basé sur une échelle de 5 couleurs : de vert foncé (catégorie des produits de très bonne qualité) au rouge (catégorie des produits les moins favorables sur le plan nutritionnel). Les lettres allant de A à E sont associées aux différentes catégories de produit pour optimiser l'accessibilité et la compréhension de Nutri-score par le consommateur.

Le score prend en compte pour 100 grammes de produit, la teneur :

- En nutriments et aliments à favoriser (fibres, protéines, fruits et légumes),
- En nutriments à limiter (énergie, acides gras saturés, sucres, sel).

Après calcul, le score obtenu par un produit permet de lui attribuer une lettre et une couleur.

Les produits concernés par ce système d'étiquetage sont tous les aliments transformés, excepté les herbes aromatiques et toutes les boissons, excepté les boissons alcoolisées.

II) Création de la base de données :

Afin d'analyser la transparence de nos différents algorithmes développés, Nous sommes amenés à créer une base de données des produits alimentaires. Pour alimenter notre base de données (fichier csv) nous avons utilisé la technique de Scrapting de données.

Dans un premier temps, nous avons récupéré les informations de 867 produits alimentaires différents et ensuite nous avons choisi 50 produits au hasard de notre fichier csv pour ne pas avoir les mêmes donnéess que les autres groupes.

Nous avons récupéré nos données sur le site de l'association Open Food Fact.

III) Fonction d'utilité additive

Définition

Pour chaque critère d'un aliment nous allons appliquer une fonction qui est différentes, puis sommer les résultats de chaque fonction.

Formule :

$$F((u_1(x_1), \ldots, u_n(x_n)) = \sum_{i=1}^{n} u_i(x_i)$$

X est un aliment
X_i est le critère i d'un aliment
U_i est la fonction appliquée à un critère i

Implémentation

Pour cet algorithme il nous suffit donc de trouver la meilleure définition possible pour les U_i. Pour le fixer nous avons choisi d'utiliser la méthode présente dans le calcul du nutri-score tel qu'il est défini actuellement, pour chaque critère nous allons donc obtenir un score (ou nombre de point) positif ou négatif, une fois sommé on obtient donc le nutri-score. Le résultat est donc exactement celui du nutri-score.

Conclusion sur cet algorithme

Cet algorithme est pour nous le meilleur car ses prédictions sont toujours juste, il nous a d'ailleurs servi de base de référence pour les autres algorithmes. Pour ce qui est de la transparence il plutôt simple à comprendre la seule notion que nous devons connaitre est celle de fonction. Il remplit donc tous les critères pour être un bon algorithme de décision.

IV) Programme Linéaire :

Implémentation

Le but de cette partie de projet est de trouver une fonction d'utilité additive qui exprime le score de chaque aliment en fonction de la somme de score des différentes utilités.

Pour cela nous avons effectué un programme linéaire qui prends paramètre le fichier csv contenant les valeurs nutritionnelles des produits et retourne un DataFrame contenant les scores de produit.

Pour bien analyser la transparence de notre modèle, nous avons définit la fonction objective comme une fonction qui optimise la somme des écarts entre les différents classe produit et des écarts entre les utilités.

Pour implémenter le programme linéaire, nous avons utilisé la librairie Pulp de Python qui contient un solveur pour optimiser les programmes linéaires.

Dans un premier temps, nous avons créés des variables nécessaires pour notre programme linéaire en donnant des bornes correctes pour ces variables.

- X : la variable qui stocke la valeur de score d'un aliment et peut prendre des valeurs entre 0 et 120
- Epsilon : la variable qui stocke la valeur de l'écart entre 2 classes de produit. Les valeurs doivent être compris entre 1 et 120
- Alpha : la variable qui stocke la valeur de l'écart entre 2 utilités (composant d'un aliment). Cette variable peut prendre des valeurs entre 0.001 et 120

Nous avons choisi une borne inférieure pour la variable alpha égale à 0.001 car nous avons estimé que certains scores d'utilité peuvent avoir des valeurs très petite. En implémentant la suite de programme linéaire, nous avons constaté que le choix des valeurs de borne inférieure pour les variables est important. Par exemple en fixant cette valeur égale à 1 pour la variable alpha, le problème ne donnait pas une solution réalisable.

Nous avons également créé un dictionnaire pour stocker les variables associées aux fonctions marginales des différents aliments.

La fonction objective est la suivant : $F(x)= \sum_{i=1}^{50} alpha(i) + \sum_{i=1}^{4} epsilon\ (i)$

Et nous essayons d'optimiser cette fonction en respectant les contraintes sur les données de notre fichier csv. Nous souhaitons optimiser les écarts entre les scores de différentes classes et différentes utilités.

Vu que notre les données générées par notre programme linéaire doivent satisfaire les contraintes des données en entrée (fichier csv), alors il faut fixer des contraintes à notre PL et pour cela nous avons implémenté 3 types de contraintes :

1) Les contraintes sur le score de chaque aliment : la somme de score des utilités de chaque aliment est égale au score d'aliment i. $\sum_{i=1}^{6} u(i) = x(j)$
Création de 50 contraintes avec une contrainte par aliment

```
aliment_1_constraint: - aliment_1 + utilite_1_alim_1 + utilite_2_alim_1
 + utilite_3_alim_1 + utilite_4_alim_1 + utilite_5_alim_1 + utilite_6_alim_1
 = 0
```

2) Les contraintes de monotonie sur les classes : Chaque aliment appartenant à une classe de nutri-score meilleur par rapport à un autre aliment possède un score supérieur. Par exemple dans notre base de données aliment numéro 0 a un nutri-score A et l'aliment numéro 1 a un nutri-score B, cela vaut dire que dans les données de sortie (optimisé) on exige que le score de premier aliment soit meilleur que le deuxième.

   Pour ajouter ces contraintes, nous avons créé 5 listes aliment_A, aliment_B, aliment_C, aliment_D et aliment_E,et dans chaque liste nous ajoutons des variables X(i) qui ont des score respectives A,B,C,D,E. par la suite, on ajoute les contraintes à notre PL de telle sort que par exemple chaque variable X(i) dans la liste aliment_B a un score inférieur à ceux qui sont dans la liste aliment_A. il y a un écart Epsilon(0) entre ces 2 aliments.

```
_C1: - aliment_0 + aliment_1 + epsilon_0 <= 0

_C2: - aliment_0 + aliment_6 + epsilon_0 <= 0

_C3: - aliment_0 + aliment_10 + epsilon_0 <= 0

_C4: - aliment_0 + aliment_16 + epsilon_0 <= 0
```

3) Les contraintes de monotonie sur les critères : ici on souhaite maximiser le score des aliments donc les critère 2, 3 ou Fibre et Protéine sont favorables et doivent être maximiser et les autres critères (Énergie, Acide gras saturé, sucre et sodium) sont défavorable et ils doivent être minimiser.

   Pour les contraintes de monotonies des critère favorables, Nous avons créé une liste triée (de plus petit à plus grande) des valeurs de chaque critère à partir de notre fichier csv et nous avons ajouté les contraintes de telle sort que $i^{ème}$ utilité de critère Énergie par exemple doit avoir un score plus grande que $i+1^{ème}$ utilité.
   Pour les contraintes de monotonie des critère défavorable, nous avons fait la même démarche sauf que cette fois-ci $i^{ème}$ utilité de critère Fibre doit avoir un score plus petit que $i+1^{ème}$.

```
_C408: alpha_0 + utilite_1_alim_27 - utilite_1_alim_5 <= 0

_C409: alpha_1 + utilite_1_alim_20 - utilite_1_alim_27 <= 0

_C410: alpha_2 - utilite_1_alim_20 + utilite_1_alim_26 <= 0

_C411: alpha_3 + utilite_1_alim_1 - utilite_1_alim_26 <= 0
```

Conclusion sur cet algorithme :

Cet algorithme n'est pas très transparent pour des personnes qui ne sont pas dans le métier et n'ont pas de connaissance en informatique. L'algorithme est assez compliqué à comprendre pour un client. Après la performance de l'algorithme est plutôt bonne car nous avons constaté que pour 50 aliments, le programme linéaire trouve une solution réalisable dans un délai de 1 ms.

Voici les résultats obtenus :

```
Statut: Optimal
aliment_0 = 60.495
aliment_1 = 41.499
aliment_10 = 41.499
aliment_11 = 40.499
aliment_12 = 39.499
aliment_13 = 0.011
aliment_14 = 60.495
aliment_15 = 0.004
aliment_16 = 41.499
aliment_17 = 0.008
aliment_18 = 60.495
aliment_19 = 39.499
```

```
utilite_6_alim_45 = 0.994
utilite_6_alim_46 = 19.497
utilite_6_alim_47 = 0.0
utilite_6_alim_48 = 0.996
utilite_6_alim_49 = 0.489
utilite_6_alim_5 = 0.0
utilite_6_alim_6 = 0.0
utilite_6_alim_7 = 18.979
utilite_6_alim_8 = 0.0
utilite_6_alim_9 = 0.001
Valeur fonction objectif =  99.97800000000021
```

Le dataframe final contenant les scores des utilités de 40 aliments :

	Énergie	Fibres	Protéines	Acides gras saturés	Sucre	Sodium
0	0.500	19.993	0.508	19.999	19.493	0.002
1	1.990	19.513	0.002	0.000	19.994	0.000
2	1.988	0.000	20.000	20.000	0.005	18.502
3	0.010	0.487	20.000	20.000	0.002	0.000
4	0.012	0.001	0.489	19.996	0.001	20.000
5	20.000	0.000	20.000	0.001	0.498	0.000
6	1.989	19.014	0.495	20.000	0.001	0.000
7	0.004	20.000	0.511	0.003	0.002	18.979
8	0.002	20.000	0.512	0.002	18.983	0.000
9	0.006	0.000	0.000	0.005	0.000	0.001
10	0.513	20.000	0.517	0.971	0.000	19.498
11	0.505	0.000	0.000	19.998	19.995	0.001
12	0.009	19.000	20.000	0.488	0.000	0.002
13	0.005	0.000	0.000	0.006	0.000	0.000
14	0.502	20.000	19.993	19.998	0.001	0.001
15	0.004	0.000	0.000	0.000	0.000	0.000
16	0.511	0.000	1.489	0.000	20.000	19.499
17	0.006	0.000	0.000	0.000	0.000	0.002
18	1.985	20.000	0.004	18.987	19.518	0.001
19	0.003	20.000	0.009	0.001	0.001	19.485
20	1.992	0.000	0.007	20.000	19.499	0.001
21	1.986	0.503	0.003	18.019	19.993	0.995
22	0.008	0.000	20.000	20.000	0.489	0.002
23	1.983	19.516	0.001	0.001	19.998	0.000
24	0.508	19.985	0.000	20.000	0.003	0.003
25	0.002	0.000	0.511	0.000	0.000	0.000
26	1.991	0.000	0.000	18.506	0.002	20.000
27	1.993	20.000	0.002	0.970	20.000	17.530
28	0.003	19.992	0.005	0.000	0.000	19.499
29	0.005	0.000	0.006	0.004	0.000	0.001
30	0.000	0.000	0.510	0.000	0.000	0.003
31	0.488	0.000	0.010	19.494	1.009	19.498
32	0.001	0.000	0.000	0.000	0.000	0.000
33	0.011	19.510	0.001	20.000	0.000	0.977
34	0.510	20.000	0.987	0.001	20.000	0.001
35	19.495	0.000	0.003	19.997	0.004	0.000
36	0.507	20.000	0.023	0.968	20.000	0.001
37	1.984	19.995	0.006	18.510	20.000	0.000
38	0.007	0.504	19.984	19.999	0.004	0.001
39	0.001	20.000	20.000	19.999	0.003	0.492
40	19.494	1.508	0.000	19.495	19.997	0.001

V) Electre-Tri

La méthode ELECTRE TRI est une méthode de décision multi critère adapté aux problèmes de classification. Elle semble donc être une bonne idée de tester ces résultats quand on l'applique à notre problème pour déterminer le nutri-score de différents aliments.

Explication de l'algorithme :

Cette méthode consiste à définir différents seuilles. Il existera pour chaque critère d'un aliment 6 seuils que l'on notera b_{ij} *(i correspond au critère étudié, j correspond au numéro du seuil).* Les seuils d'un même critère sont soit classés par ordre croissant si le critère est à maximiser ou par ordre décroissant si le critère est à minimiser. Par conséquent fixer ces variables b_{ij} est une chose qui sera capitale dans notre algorithme. Ensuite on va déterminer pour chaque critère la variable C_{ij} *(i correspond au critère étudié, j correspond au numéro du seuil).*

Pour les critères à maximiser on a :

$$C_{i_j} = \begin{cases} 1, & Valeur\ du\ critère < b\_i \\ 0, & Valeur\ du\ critère \geq b\_i \end{cases}$$

Pour les critères à minimiser on a :

$$C_{i_j} = \begin{cases} 0, & Valeur\ du\ critère < b\_i \\ 1, & Valeur\ du\ critère \geq b\_i \end{cases}$$

Ensuite on détermine les C_i grâce à la formule suivante :

$$C_i = \frac{\sum_{j=1}^{n} C_i}{\sum_{j=1}^{n} k_j}$$

Les k_j représentent des poids
Ensuite on cherche le premier C_i qui vérifie la condition suivante :

$$C_i \geq \lambda$$

Le premier qui vérifie cette condition détermine sa classe.

Détaille de nos travaux

La plus grosse difficulté pour cet algorithme est donc déterminée les b_{ij} et les λ qui nous permettrons d'obtenir les meilleurs résultats. Il est évident qu'au vu du trop grand nombre de choix il est impossible de faire le test pour toutes les possibilités. Il faut donc trouver un moyen de les déterminer de façon logique. Pour ce faire nous avons choisi trois méthodes différentes :

- Utiliser les seuils qui sont décrit dans le nutri-score pour chaque critère
- Utiliser la moyenne (extraite de notre dataset et l'utiliser) pour déterminer leur valeur
- Utiliser la valeur maximum (extraite de notre dataset et l'utiliser) pour déterminer leur valeur

Le premier fu un échec, ce qui semble tout à fait normale car ces seuils ont été fait pour un autre algorithme que le nôtre et que ce dernier ne fonctionne pas de la même manière que ELECTRE TRI. Pour la deuxième nous avons remarqué une légère amélioration. Le problème de cette méthode est qu'elle prend compte de comment sont répartie les valeurs pour les critères, (c'est-à-dire que si sur notre data set la moyenne des fibres est élevé sur notre dataset on aura tendance à toujours sous-évalué le taux de fibres). On pense cependant que cette approche pourrait marcher avec un dataset qui serait beaucoup plus volumineux et qui prendrais en compte la variance et l'écart types. Malheureusement on perdrait beaucoup en transparence pour notre algorithme et cela demanderait beaucoup de tempe d'analyse sur nos données, c'est pour ces raison que nous avons choisi de passer à une autre solution. La troisième méthode obtient des résultats beaucoup plus intéressants que les deux précédentes, cependant il faut faire attention que le maximum que l'on choisit pour nos critères ne soit pas une valeur excessivement grande, c'est pourquoi nous avons enlevé certaine valeur de notre dataset. Les résultats de cette méthode son très bon pour les notes basse (un nutriscore de A) mais décroit à chaque rang. Nous avons donc choisi de garder cette méthode en cherchant à améliorer notre score. Pour ce qui est des λ nous n'avons pas trouvé de façon normale de les déterminer nous avons donc choisi de les fixer en faisant des tests et voir si on obtenait de meilleurs résultats.

Finalement nous n'avons pas réussie à régler ce problème lié à notre choix des b_{ij} car le problème venait du fait qu'il y avait une compensation entre les critères (c'est-à-dire qu'une très mauvaise note sur un critère pouvait être compensé par les autres, c'est le cas notamment du Nutella). Nous avons essayé d'implémenter des seuils de veto mais toutes ces tentatives donnaient des plus mauvais scores de prédiction, et par manque de temps nous n'avons pas pu pousser au bout tous nos tests, il est donc probable que cela aurait pu être meilleure.

Conclusion sur cet algorithme :

Pour conclure sur cet algorithme nous pouvons donc dire qu'il n'a pas une grande performance mais qu'il est très probable d'obtenir de bien meilleur résultat en ajoutant des bons seuils de veto ou en trouvant une meilleure manière de déterminer nos seuils. Pour ce qui de la transparence il n'est pas très facile à comprendre pour une personne qui n'est pas familière à la programmation, c'est pourquoi nous pensons que ce n'est pas le meilleur choix.

Cette méthode consiste à définir différents seuilles. Il existera pour chaque critère d'un aliment 6 seuils que l'on notera b_{ij} *(i correspond au critère étudié, j correspond au numéro du seuil).*

Les seuils d'un même critère sont soit classés par ordre croissant si le critère est à maximiser ou par ordre décroissant si le critère est à minimiser. Par conséquent fixer ces variables b_{ij} est une chose qui sera capitale dans notre algorithme. Ensuite on va déterminer pour chaque critère la variable C_{ij} *(i correspond au critère étudié, j correspond au numéro du seuil).*

VI) Arbre et Forêt

Définition

Un arbre de décision est un outil d'aide à la décision représentant un ensemble de choix sous la forme graphique d'un arbre. Les différentes décisions possibles sont situées aux extrémités des branches (les « feuilles » de l'arbre), et sont atteintes en fonction de décisions prises à chaque étape. L'arbre de décision est un outil utilisé dans des domaines variés tels que la sécurité, la fouille de données, la médecine, etc. Il a l'avantage d'être lisible et rapide à exécuter. Il s'agit de plus d'une représentation calculable automatiquement par des algorithmes d'apprentissage supervisé.

Implémentation

Pour cet algorithme nous avons décidé d'utiliser la bibliothèque ***Scikitlearn.*** Celle-ci nous permettre de de créer, modifier nos arbres plus facilement. Nous appliquons l'algorithme de CART pour générer nos arbres. Les arbres que nous avons créés ne donnais pas de très bon résultat au début mais devenais bien meilleure quand nous augmentions la profondeur de nos arbres. Nous avons aussi grâce à cette même bibliothèque créé une forêt d'arbres qui consiste à avoir plusieurs arbres de décision différent et à les faire voter entre eux pour savoir le score.

Conclusion

Pour ce qui est de cet algorithme nous avons plusieurs nous avons de très bon résultat si on augmente assez la profondeur pour un arbre seul. Pour ce qui est de la forêt le score est un peu meilleur que celui de l'arbre seul mais il n'est pas nécessaire de faire plus de 10 arbres car cela ne change rien au résultat avec plus. En dehors des quelques erreurs que peut faire cet algorithme il pose un gros problème sur la transparence. En effet les arbres que nous avons sont incompréhensible et trop grand même pour nous. Ces pourquoi nous n'avons pas chercher à améliorer plus que de nécessaire cette méthode