Université de Reims
Champagne-Ardenne

PRÉDICTION DU NUTRI-SCORE

RAPPORT FINAL

Master SEP
Promotion 2023/2024

Brunet Alexandre
Ertas Elif
Jupin Manon
Gabet Léo
Kpadonou Carlos
Otogondoua Ememaga
Jordhy Jean Jaurès

SOMMAIRE

Remerciements 3

Introduction 4

1. Interfaces utilisateur 6

1.1. Interface d'entrée 6

1.2. Interface de sortie 8

2. Préparation des données 12

2.1. Lien VBA-Python 12

2.2. Nettoyage des données pour l'algorithme de prédiction 13

3. Modèles utilisés 14

3.1. Régression logistique ordinale 14

3.2. Random Forest 17

Conclusion 25

Table des figures 26

Table des tableaux 27

REMERCIEMENTS

Nous tenons à exprimer notre sincère gratitude envers toutes les personnes qui ont contribué à l'achèvement de ce projet. Leur expertise, encouragement et retours constructifs tout au long du processus ont été très précieux pour construire un projet sérieux et pertinent.

Nous tenons d'abord à exprimer notre gratitude envers Morgan Cousin pour son apport significatif en termes de méthodes et d'outils de travail. Ses retours constructifs ont joué un rôle déterminant dans notre amélioration, que ce soit au niveau de nos compétences orales ou de notre organisation. Nos remerciements vont également à Amor Keziou pour ses avis en termes de choix de modèles de prédiction qui nous ont permis d'avoir un regard critique sur les modèles que nous avons utilisés. Enfin, nous gratifions Arona Diene pour ses conseils et son aide dans la construction et le développement de notre interface ainsi que du lien entre VBA et Python.

INTRODUCTION

Au cours des dernières années, l'émergence croissante des préoccupations environnementales s'est étendue à la question de la santé et du bien-être physique par le biais de l'alimentation. La convergence de ces deux aspects, combinée au désir d'informer le consommateur, a incité l'État à établir une norme d'étiquetage pour les produits transformés connue sous le nom de Nutri-Score. Cette norme classe les produits en fonction de leur impact positif ou négatif sur la santé des consommateurs, symbolisé par des lettres de A (très bon pour la santé) à E (très mauvais pour la santé). Ce classement repose sur les composants nutritionnels des produits.

C'est dans ce contexte que notre application se concentre sur la prédiction du Nutri-Score des produits à partir de leurs valeurs nutritionnelles. Pour en savoir plus sur cette application, veuillez consulter notre dépôt GitHub.

Cette application s'adresse à deux groupes distincts. Le premier groupe comprend les consommateurs qui souhaitent vérifier le Nutri-Score des produits ou découvrir le Nutri-Score d'un produit qui n'en dispose pas. Le second groupe est constitué de producteurs cherchant à évaluer rapidement et sans coût le Nutri-Score de leurs produits. Cette distinction a été établie pour atteindre un public plus vaste, mais aussi et surtout parce que certaines informations nécessaires ne sont accessibles qu'aux producteurs.

Pour la mise en œuvre de cette application, nous nous sommes appuyés sur une base de données regroupant plus de 52 000 produits, chacun avec ses caractéristiques nutritionnelles, les variables explicatives et leur Nutri-Score (voir tableau 1).

Variables*	Description (quantité pour 100g)
energy	Kilocalories
fat	Matières grasses
saturated-fat	Graisses saturées
trans-fat	Graisses transformées
cholesterol	Cholestérol
carbohydrates	Glucides
sugars	Sucres
fiber	Fibres
proteins	Protéines
salt	Sel
sodium	Sodium
vitamin-a	Vitamine A
vitamin-c	Vitamine C
calcium	Calcium
iron	Fer
nutrition_grade_fr	**Notre variable cible qui est décrite par des lettres allant de A à E exprimant la qualité nutritionnelle du produit.**

Tableau 1 : Description de toutes les variables utilisées. * _100g

Cette base est présente sur le site de l'open data du gouvernement qui rassemble les produits présents sur le site Open Food Facts qui met à jour régulièrement ses données. Grâce à cela, nous allons mobiliser des outils d'apprentissage statistique pour qu'un consommateur ou producteur, en entrant les informations du produit, puisse évaluer de la pertinence du Nutri-Score d'un produit (ou alors connaître le son Nutri-Score dans le cas où il n'y a pas accès).

1. Interfaces utilisateur

1.1. Interface d'entrée

Lorsque l'on pose le premier pas sur notre application, une page d'accueil s'affiche (figure 1), introduisant notre application en présentant son public cible, les motivations derrière son développement, ainsi qu'un avertissement destiné aux utilisateurs. Ce message encourage ces derniers à entrer les données de manière spécifique, en pensant en termes de 100g, c'est-à-dire en considérant les valeurs comme celles figurant sur les emballages de produits que l'on trouve habituellement en supermarché. En outre, la page d'accueil propose différents historiques et tableaux de bord, ainsi qu'un message spécialement conçu pour les consommateurs. Enfin, un bouton est mis à disposition pour ouvrir le formulaire dans lequel les utilisateurs pourront saisir les valeurs nécessaires.

Figure 1 : Page d'accueil.

Poursuivons avec la création d'une interface utilisateur sous forme d'un formulaire dans Excel qui a plusieurs parties (figure 2). La première, la plus importante, est celle du choix entre consommateur et producteur. Ce choix se justifie dans le fait que certaines informations ne se

trouvent pas sur les paquets des produits (comme le cholestérol), ou en tout cas pas à tous les coups. Ensuite, nous avons le nom du produit que l'utilisateur doit insérer. Nous avons également la partie de l'historique : l'utilisateur peut visualiser les différents produits qu'il a déjà inséré. Enfin, nous avons la plus grande partie qui concerne les informations que l'utilisateur veut insérer.

Dans le cas où un produit serait déjà présent dans l'historique et que l'utilisateur insère à nouveau le nom de ce produit, un message s'afficher en disant que le produit existe déjà et donne également le choix de modifier ou non les informations de ce produit. Aussi, lorsque l'utilisateur fait le choix entre consommateur et producteur (choix qu'il doit faire en premier, sans quoi aucune information ne peut être insérée), les informations grisées sont celles qui ne sont pas disponibles pour le type choisi. La raison de cette sécurité tient dans le modèle statistique. En effet, nous avons généré 2 modèles différents, basés sur des variables différentes. Cela s'exprime sur le formulaire par la disponibilité ou non d'insérer certaines valeurs. En parlant de ces valeurs, puisqu'elles sont exprimées par 100g, aucune valeur ne doit excéder 100g (sauf pour les kilocalories).

Naturellement, nous trouvons le bouton de confirmation (« Calculer Nutri-Score ») de couleur verte qui va calculer le Nutri-Score avec les informations insérées, puis le bouton pour quitter le formulaire, en rouge. Lorsque l'utilisateur veut confirmer ses informations grâce au bouton vert, un récapitulatif ainsi qu'une demande de (re)confirmation apparaissent. De plus, le bouton vert ne sera actif que lorsque toutes les variables seront entrées dans le formulaire.

Pour finir, la vérification des valeurs est importante pour Python dans sa lecture des données des entrées. Par exemple, le formulaire supprime automatiquement un caractère, car nous attendons des valeurs numériques au format « 0.0 ». Ainsi, si « 0,0 » est saisi, notre code le met à jour avec le format « 0.0 », le seul format numérique accepté en langage Python. Cela nous a permis d'éviter d'afficher des messages à chaque case, car nous trouvons ce principe très lourd pour l'utilisateur. Avec notre code, les entrées et corrections sont plus fluides et l'utilisateur peut poursuivre son activité, sans à se soucier des détails comme celui-ci.

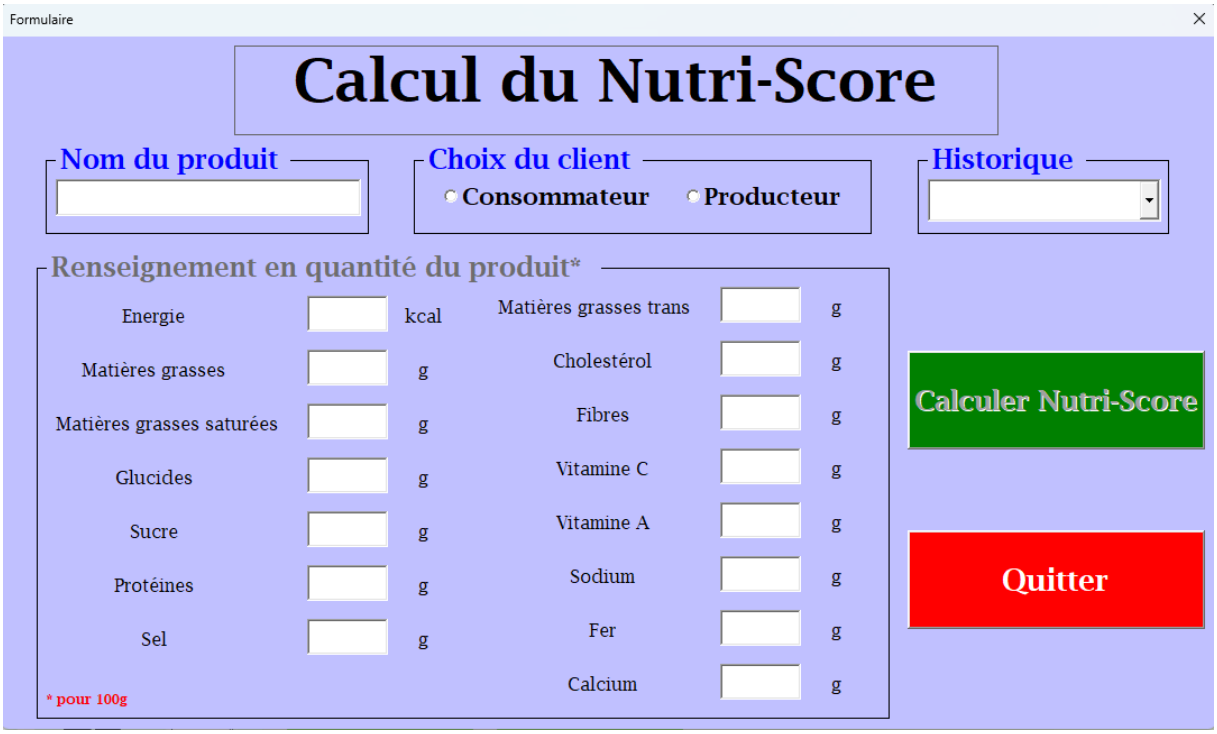

Figure 2 : Formulaire d'entrée.

1.2. Interface de sortie

Quand l'utilisateur a réussi à entrer les valeurs de son produit, le Nutri-Score s'obtient par la prédiction de notre algorithme exécuté par un script Python, via l'API. En fonction de ce que l'utilisateur à cocher, il serait dirigé vers le dashboard du producteur (figure 3) ou celui du consommateur (figure 4) dans lesquels les messages sont personnalisés en fonction du public.

À l'intérieur de ces dashboard, nous trouvons le plus important : le Nutri-Score que retourne l'algorithme. Évidemment, l'algorithme ne fait pas des choix catégoriques, mais estime des probabilités d'appartenance. C'est le Nutri-Score qui obtient la plus grande probabilité qui est mis en avant avec une jauge et un message qui commente le résultat. Quant aux autres probabilités, ils sont résumés dans le diagramme circulaire situé en dessous de la jauge qui résume toutes les probabilités. L'intérêt est de voir si l'algorithme est plutôt « confiant » dans son résultat. Par exemple, un produit qui a 68% de chance d'être classé dans la catégorie B, veut dire qu'il y a peu de doute sur son classement. Ensuite, dans la même partie des dashboard, nous avons le top 3 des nutriments qui influencent le plus le calcul des probabilités et les performances des modèles (nous en parlerons en détails dans la partie sur les modèles utilisés). À droite du dashboard, nous avons l'erreur de classification des modèles utilisés, indiquant ainsi la « fiabilité » de celui-ci. Enfin, quelques recommandations apparaissent, à la

fois sur la composition des produits des producteurs (pour qu'ils fassent attention) et sur les limites que pose l'OMS pou avertir les consommateurs des « risques » nutritionnels.

Hors du dashboard, un historique apparaît de la même façon que pour le formulaire d'entrée, ainsi, si l'utilisateur veut revoir le résultat de ses anciens produits, il n'est pas obligé de revenir sur le formulaire. L'objectif est bien de fluidifier l'expérience de l'utilisateur.

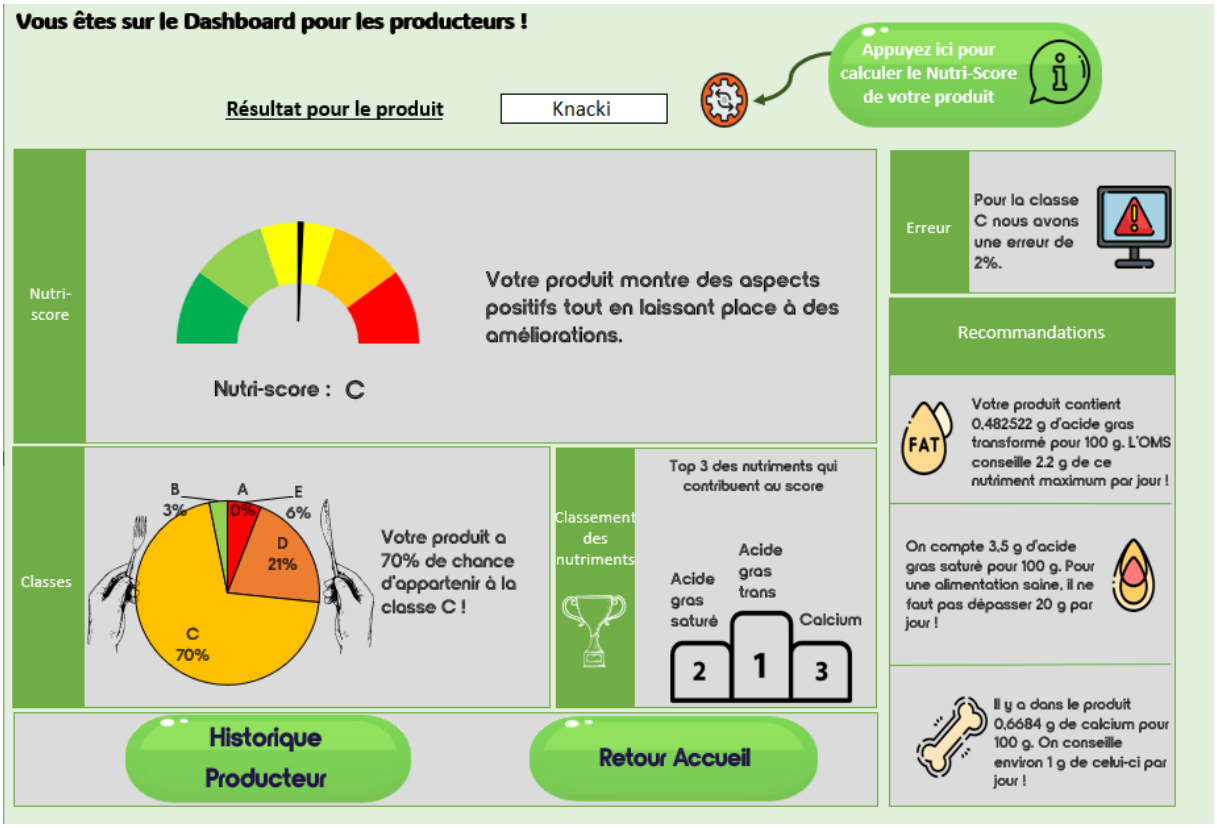

Figure 3 : Dashboard du producteur.

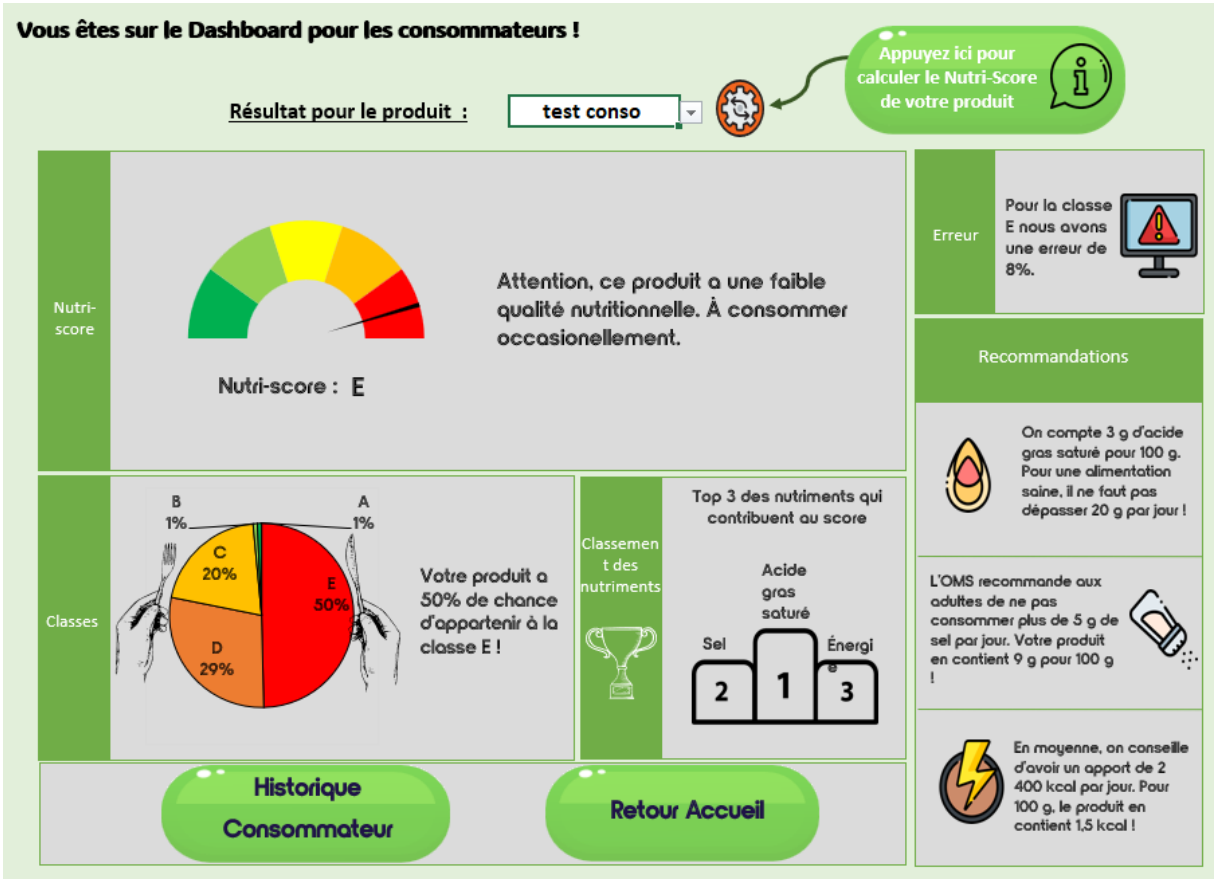

Figure 4 : Dashboard du consommateur.

En outre, nous avons mis à disposition des boutons lui permettant de revenir à l'accueil et de consulter l'historique (figure 5 et 6). Dans ces historiques, nous retrouvons toutes les informations insérer par l'utilisateur dans sa page dédiée (consommateur ou producteur) avec les différentes probabilités ainsi que le Nutri-Score le plus probable pour chaque produit. Deux boutons sont également mis à disposition : le retour à l'accueil et la visualisation du dashboard. L'utilité de tous ces boutons, c'est d'améliorer l'expérience de l'utilisateur en lui évitant les désagréments induit par la navigation entre les différentes feuilles du fichier Excel.

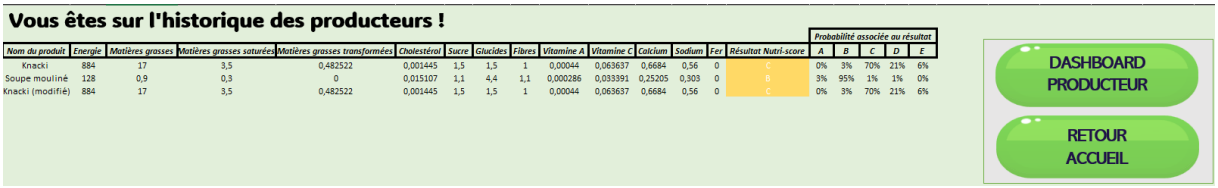

Figure 5 : Historique du producteur.

.

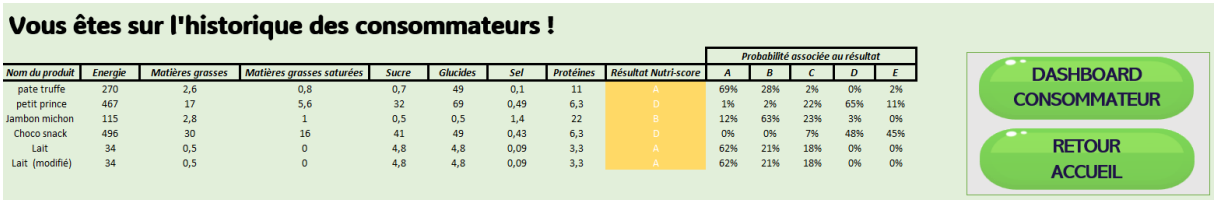

Figure 6 : Historique du consommateur.

2. Préparation des données

2.1. Lien VBA-Python

Notre application a pour objet d'utiliser à la fois VBA (pour la création d'interfaces), et à la fois Python (pour le développement d'un algorithme de prédiction). Or, ce sont deux logiciels qui ne sont pas par nature liés. C'est pourquoi nous devons faire la liaison nous-même. Certes, nous avons développé du code VBA pour la création du formulaire, etc. Mais nous avons également fait en sorte que le code VBA gère le stockage des valeurs dans un historique et qui les envoie à Python pour appliquer le modèle (selon si l'utilisateur à cocher consommateur ou producteur). Il reste maintenant à savoir comment ce lien a pu s'établir. En fait, c'est grâce à une API. La relation Python-VBA s'effectue selon le processus suivant :

1. Les utilisateurs saisissent des données dans un formulaire Excel.
2. Les données saisies sont stockées dans des variables VBA pour une manipulation ultérieure.
3. Ces variables VBA sont utilisées pour créer une structure de données JSON (format {"clé": valeur}).
4. Une API Python a été développée pour réaliser la prédiction du Nutri-Score. Cette API est hébergée sur pythonanywhere.
5. Le script de l'API extrait les deux modèles choisis contenus dans les fichiers pickle
6. Puis récupère les données issues de la requête reçue
7. Effectue la prédiction des probabilités d'appartenance à chaque groupe
8. Enfin, ces probabilités sont stockées dans des variables et mis au format JSON pour être retourné à Excel par l'API
9. Les données au format JSON sont envoyées à cette API Python pour effectuer la prédiction.
10. L'API Python renvoie une réponse au format JSON contenant les probabilité d'appartenance à chaque groupe de Nutri-Score.
11. Une fonction appelée "jsonconverter" est utilisée pour interpréter le format JSON retourné et extraire la valeur du Nutri-Score prédit.

Mais pour cela, il est nécessaire d'avoir, en plus de Microsoft Excel, Reference VBA project Microsoft scripting Runtime ainsi qu'une connexion internet, sans quoi l'API ne peut fonctionner et que donc aucune prédiction ne peut s'effectuer.

2.2. Nettoyage des données pour l’algorithme de prédiction

Le nettoyage de nos données a été relativement rapide et simple. En effet, notre base, en plus des valeurs nutritionnelles des produits, contenait d’autres informations comme la présence d’additifs par exemple. Cependant, étant donné que le Nutri-Score ne prend en compte que les valeurs nutritionnelles, nous avons décidé de toute information ne relatant pas des variables que l’on étudiait. De plus, ces autres informations étaient inexploitables, car elles manquaient cruellement de sens.

Nous avons réussi à rendre notre base de données entièrement exploitable par nos algorithmes. Cependant, une étape cruciale restait à accomplir : traiter les données manquantes. C'est la raison pour laquelle nous avons choisi de retirer les produits pour lesquels des informations étaient absentes. Finalement, nous avons pu conserver environ 29 000 produits, ce qui équivaut à un peu plus de la moitié de notre base initiale, qui comptait environ 52 000 éléments.

Par ailleurs, nous avons pris soin d'équilibrer nos données afin qu'aucune classe ne soit sur-représentée par rapport aux autres. En effet, la classe la moins représentée compte désormais 19% des données, tandis que la classe la mieux représentée en compte 22%, assurant ainsi une répartition équitable des informations dans l'ensemble de notre base de données.

3. Modèles utilisés

3.1. Régression logistique ordinale

La régression logistique ordinale, souvent désignée sous le terme de « probit ordonné » ou « logit ordonné », constitue une extension significative de la régression logistique binaire, adaptée au contexte où la variable dépendante (cible) présente une structure ordonnée. Dans notre étude, notre variable cible se décline en 5 modalités, créant ainsi 4 seuils qui délimitent les différentes catégories.

Le modèle de régression logistique ordinale peut être formulé de deux manières distinctes : le modèle probit ordonné et le modèle logit ordonné. Ces deux approches partagent un objectif commun, celui de modéliser les probabilités conditionnelles des différentes catégories de la variable dépendante en fonction des variables explicatives. Dans le cas du modèle probit ordonné, la probabilité conditionnelle que la variable dépendante soit inférieure ou égale à une catégorie k par rapport à une catégorie de référence est exprimée par la fonction de répartition normale cumulative (Φ). Soit $(Y \leq k|X)$ la probabilité que la variable dépendante Y soit dans la catégorie j ou une catégorie inférieure compte tenu des valeurs des variables explicatives X ; α_k le terme d'interception spécifique associé à la catégorie k et β le vecteur des coefficients des variables explicatives :

$$(Y \leq k|X) := \Phi(\alpha_k + X\beta)$$

D'un autre côté, le modèle logit ordonné formule ces probabilités de la manière suivante, utilisant la fonction logistique. Soit $P(Y = k|X)$ la probabilité que la variable dépendante Y soit exactement dans la catégorie k compte tenu des valeurs des variables explicatives X :

$$P(Y = k|X) := \frac{e^{\alpha_k + X\beta}}{1 + \sum_{i=1}^{K-1} e^{\alpha_i + X\beta}}$$

Ainsi, nous avons préféré conserver le modèle logit, car il utilise la fonction logistique, tandis que le modèle probit repose sur la fonction de répartition de la loi normale standard, ce qui est très contraignant. En effet, le modèle probit constitue une base théorique complexe et une hypothèse de normalité souvent trop contraignante dans les applications économétriques, comme la nôtre.

Ensuite, nous devions estimer les paramètres pour résoudre notre modèle. Cela a pu se faire grâce à la fonction de log-vraisemblance. L'objectif était de maximiser cette fonction :

$$\ell(\alpha, \beta|Y, X) := \sum_{i=1}^{N} log[P(Y_i|X_i; \alpha, \beta)]$$

Cela nécessite des algorithmes itératifs tels que Newton-Raphson, le gradient conjugué ou la méthode de Basin-hopping, étant donné l'absence de solution analytique. Lors de la maximisation de la log-vraisemblance, les algorithmes itératifs convergent vers l'estimateur du maximum de log-vraisemblance, éliminant le risque de convergence vers un maximum local. Cependant, des problèmes peuvent survenir lorsque les modalités de Y sont parfaitement séparées selon les valeurs de X. Si les données sont complètement ou quasi-complètement séparées, l'estimateur du maximum de la log- vraisemblance n'existe pas. Dans la plupart des cas concrets, les données ne sont généralement pas séparées, justifiant l'utilisation d'algorithmes itératifs pour converger vers l'estimateur du maximum de la log-vraisemblance.

Pour surmonter cette contrainte, nous avons exploré une alternative pour la sélection de variables. Nous avons opté pour un modèle probit obtenu avec l'algorithme itératif de Broyden-Fletcher-Goldfarb-Shanno, présentant des paramètres similaires à notre modèle de base. Bien que ce modèle n'ait pas convergé, il a fourni des indicateurs AIC et BIC du modèle. Nous avons ensuite procédé à une sélection de variables par recherche backward pour minimiser les critères AIC et BIC. Enfin, pour les deux modèles obtenus selon chaque critère, nous avons évalué l'erreur théorique de classification par validation croisée k-fold avec $k = 5$.

L'erreur théorique de classification par validation croisée k-fold mesure la performance d'un modèle en termes de classification, en prenant en compte la moyenne des erreurs sur k itérations de validation croisée. La validation croisée est une technique permettant de diviser le jeu de données en k sous-ensembles (plis) et d'itérer sur ces plis pour évaluer la performance du modèle.

Mathématiquement, l'erreur de classification peut être formulée comme suit. Supposons que Y_i soit la vraie classe de l'observation i et $\hat{Y}_i$ soit la classe prédite par le modèle pour cette observation. Alors, l'erreur de classification pour une itération de validation croisée k-fold est donnée par :

$$Erreur\ de\ classification_k := \frac{1}{N_k} \sum_{i \in Plis_k} I(Y_i \neq \hat{Y}_i)$$

Où N_k est le nombre total d'observations dans le pli k ; $Plis_k$ est l'ensemble des indices des observations dans le pli k et $I(\cdot)$ est la fonction indicatrice qui prend la valeur 1 si la condition est vraie et 0 sinon. La moyenne des erreurs de classification sur toutes les itérations de validation croisée donne l'erreur théorique de classification par validation croisée k-fold avec $k = 5$:

$$Erreur\ de\ classification\ moyenne = \frac{1}{5}\sum_{k=1}^{5} Erreur\ de\ classification_k$$

Les critères de l'AIC (Critère d'Information Akaike) et du BIC (Critère d'Information Bayésien) prennent en compte la qualité du modèle en termes de vraisemblance et la pénalisation en fonction du nombre de paramètres (de variables). Cependant, le BIC pénalise d'autant plus les modèles complexes que l'AIC. Ces deux critères sont définis comme ceci :

$$AIC(M_k) := -2 \cdot \log\left[l(\alpha, \beta|Y, X)\right] + 2k$$

$$BIC(M_k) := -2 \cdot log(l(\alpha, \beta|Y, X)) + k \cdot log(N)$$

Où k est le nombre de variables dans le modèle et N est le nombre d'observations. Pour trouver les modèles optimaux au sens de ces deux critères, nous avons calculé quel sont les modèles qui minimisent les deux critères :

$$k^*_{AIC} := \underset{k\in\{1,\dots,m\}}{\arg\min}\ AIC(M_k)$$

$$k^*_{BIC} := \underset{k\in\{1,\dots,m\}}{\arg\min}\ BIC(M_k)$$

Enfin, pour élargir le choix du modèle optimal, nous avons également utilisé les méthodes de régularisation Ridge et Lasso. Ils sont utilisés pour résoudre le problème de multicollinéarité et pour sélectionner les variables importantes. Dans les deux cas, ces méthodes de régularisation peuvent aider à prévenir le sur-ajustement du modèle et à sélectionner les variables importantes tout en ajustant les modèles. La régression Ridge et Lasso introduisent respectivement une pénalité L2 et une pénalité L1 sur les coefficients du modèle. Les fonction objective à optimiser sont les suivantes :

$$Minimiser\ J(\alpha, \beta) := \sum_{i=1}^{N} \log\left[P(Y_i|X_i, \alpha, \beta)\right] + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

$$Minimiser\ J(\alpha,\beta) := \sum_{i=1}^{N} log\ [P(Y_i|X_i,\alpha,\beta)] + \lambda_1 \sum_{j=1}^{p} |\beta_j|$$

Avec N est le nombre d'observations, Y_i est la variable dépendante pour l'observation i, X_i est le vecteur des variables explicatives pour l'observation i, α sont les seuils de catégorie, β sont les coefficients des variables explicatives, et λ_i pour tout $i \in \{1,2\}$ étant le paramètre de régularisation L1 ou L2. Le choix de ce dernier est crucial pour garantir que les modèles obtenus présentent une erreur minimale.

Finalement, pour choisir le modèle qui classifie le mieux en fonction des variables que dispose le producteur, nous avons comparé tous les modèles optimaux et le modèle initial (tableau 2). C'était ce dernier que nous avions décidé de retenir.

Modèles optimaux	Initial	Au sens de l'AIC	Au sens du BIC	Ridge	Lasso
Erreur de classification (en %)	26,1	28	28	43,5	27,5

Tableau 2 : Comparaisons des erreurs de classification des différents modèles optimaux.

Pour conclure sur ce modèle, étant donné les résultats sur seulement le modèle du producteur, nous avons fait le choix de ne pas davantage développer le modèle de régression logistique ordinale. Nous avons trouvé un autre modèle bien plus performant que nous avons bien développé cette fois : le Random Forest.

3.2. Random Forest

Étant donné l'erreur relativement élevée qu'obtient la régression logistique ordinale, nous avons décidé de nous tourner vers une autre méthode de classification : le Random Forest. Globalement, ce modèle est une collection d'arbres de décision aléatoires décorrélés qui est beaucoup moins sensible aux données d'entraînement et qui est également beaucoup plus précis dans ses prédictions (car moins de chance de sur-ajustement).

Le Random Forest s'appuie sur la méthode du bagging (bootstrap aggregating). Elle consiste à entraîner aléatoirement nos arbres de décision sur une partie des données et une partie des variables sur le jeu de données d'entraînement, ce qui donnera des échantillon bootstrap. Pour chaque échantillon bootstrap, un arbre de décision est créé. Chacun divise les données en

fonction des caractéristiques qui maximisent la pureté des nœuds. La pureté peut être mesurée grâce à l'indice de Gini :

$$Gini(t) := 1 - \sum_{i=1}^{C} p(i|t)^2$$

Avec C le nombre de classes (5 dans notre cas) et $p(i|t)$ le proportion d'observation de la classe i dans le nœud t. Plus précisément, nous calculons cette proportion ainsi :

$$p(i|t) := \frac{Nombre\ d'observation\ de\ la\ classe\ i\ dans\ le\ noeud\ t}{Nombre\ total\ d'observation\ dans\ le\ noeud\ t}$$

Chaque arbre de décision, qui peut être différent ou similaire, va voter. Pour la classification du Random Forest, la prédiction est basée sur le vote majorité des arbres de décision :

$$Prédiction\ du\ Random\ Forest := argmax(\sum_{i=1}^{N} Prédiction\ de\ l'arbre_{i,c})$$

Avec N est le nombre d'arbres dans le Random Forest et $Prédiction\ de\ l'arbre_{i,c}$ est la prédiction de la classe c pour l'arbre i.

Maintenant, une question est encore en suspens : combien d'arbre nous devons avoir dans notre forêt et quelle doit être leur profondeur afin d'obtenir de meilleures performances ? Pour répondre à cette question, nous devons chercher ces hyperparamètres de façon appropriée. Autrement dit, nous devons trouver un nombre d'arbres et une profondeur appropriés qui permettent de maximiser la précision du modèle (et donc minimiser l'erreur de classification). La précision globale du Random Forest est exprimée ainsi :

$$Précision := \frac{\sum_{i=1}^{C} C_{i,i}}{N}$$

Avec $C_{i,i}$ les observations correctement classées.

Pour ce faire, une technique habituelle consiste à définir une grille de valeurs possibles pour les hyperparamètres, puis de spécifier le nombre d'itérations ou d'évaluations à effectuer de manière aléatoire. L'algorithme sélectionne ensuite des combinaisons aléatoires d'hyperparamètres à partir de cette grille et évalue les performances du modèle avec chaque

combinaison grâce à une validation croisée. À la fin du processus, les hyperparamètres produisant les meilleures performances sont identifiés, offrant ainsi une approche plus rapide et efficace pour l'optimisation des paramètres de Random Forest. Cette approche nous a permis de trouver rapidement des combinaisons d'hyperparamètres qui maximisent les performances de nos modèles tout en évitant la lourdeur d'une recherche exhaustive. Maintenant, avec les hyperparamètres optimisés en main, on peut maintenant présenter les performances obtenues sur les données de test pour nos deux modèles (figures 7 et 8).

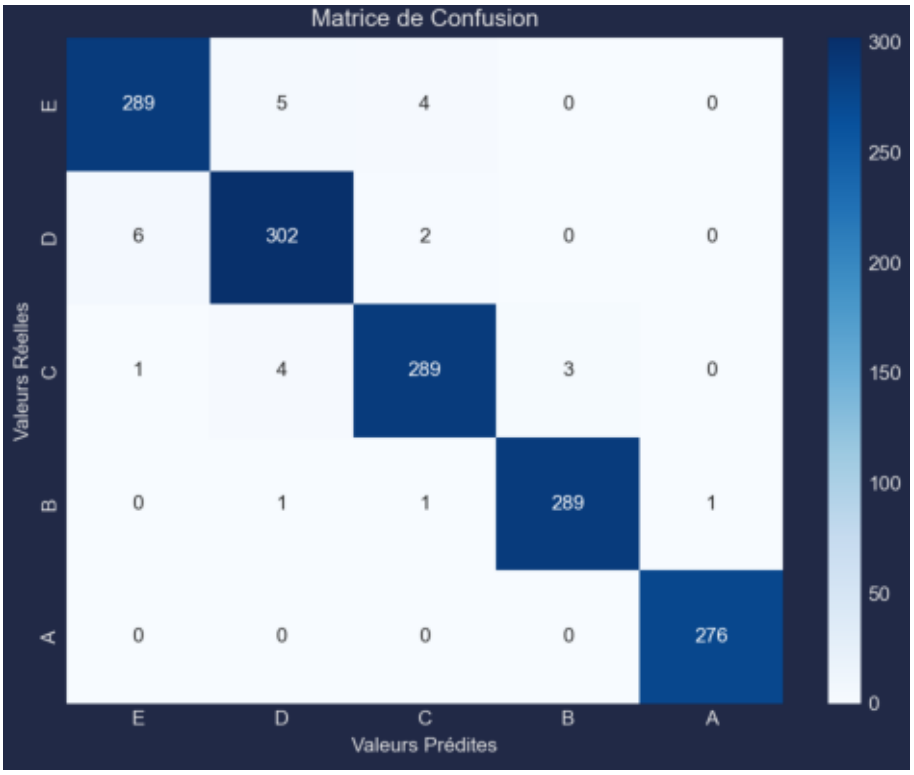

Figure 7 : Matrice de confusion pour le modèle des producteurs. **Lecture** : 276 produits ont été correctement classés dans le Nutri-Score A contre un produit qui a été classé dans le A alors qu'il était noté B.

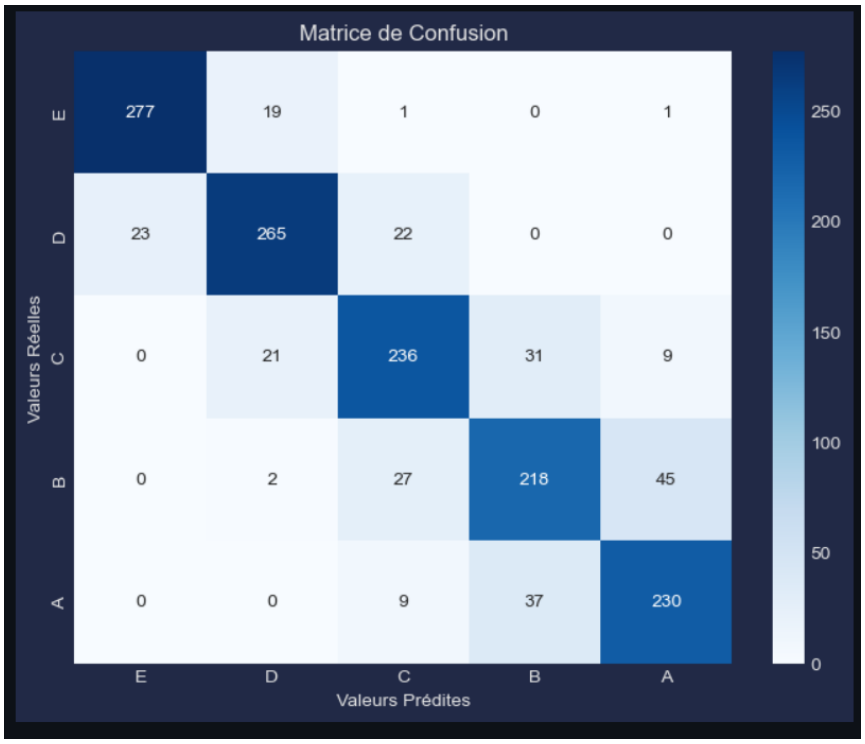

Figure 8 : Matrice de confusion pour le modèle des consommateurs. **Lecture** : 236 produits ont été correctement classés dans le Nutri-Score C contre 21 produit qui a été classé dans le D alors qu'il était noté C et 31 produits notés C mais qui ont été classé B.

Avec ces deux différentes matrices, nous pouvons établir un tableau résumant l'erreur globale des 2 modèles ainsi que l'erreur de chaque classe à partir des données de test (tableau 3).

Modèles	A	B	C	D	E	Moyenne
Producteur	1	1	2	4	2	2
Consommateur	18	24	20	14	8	17

Tableau 3 : Erreur des modèles après évaluation du Random Forest sur la base de test. **Lecture** : Un produit classé A par le modèle de Random Forest sur les caractéristiques du producteur a une chance de 1% d'être mal classé.

Afin de tester la robustesse des hyperparamètres sélectionnés, on a mis en place une approche itérative. Dans cette démarche, on entraîne le modèle avec les hyperparamètres optimisés en utilisant une base d'entraînement équilibrée pour toutes les classes. Cette base d'entraînement est tirée aléatoirement de la base de données globale à chaque itération. De manière complémentaire, on constitue une base de test également tirée aléatoirement de la base globale, sans nécessairement maintenir un équilibre entre les classes. Ce processus itératif a été répété plusieurs fois pour observer la valeur moyenne vers laquelle l'erreur de classification

converge. Cette approche permet de garantir que les performances du modèle ne sont pas simplement le résultat d'une bonne adaptation à une configuration spécifique des données, mais qu'elles demeurent stables et fiables sur des ensembles de données variés. Les résultats obtenus à partir de cette validation itérative renforcent la confiance dans la capacité du modèle à généraliser de manière efficace à de nouvelles données.

Cette procédure a été effectuée à la fois sur le modèle producteur et du consommateur (figure 9 et 10). Et grâce à la loi des grands nombre, nous voyons que la moyenne des erreurs est très proche (dans le cas du modèle du consommateur), voire égale (dans le cas du modèle du producteur) à ce que l'on a obtenu précédemment.

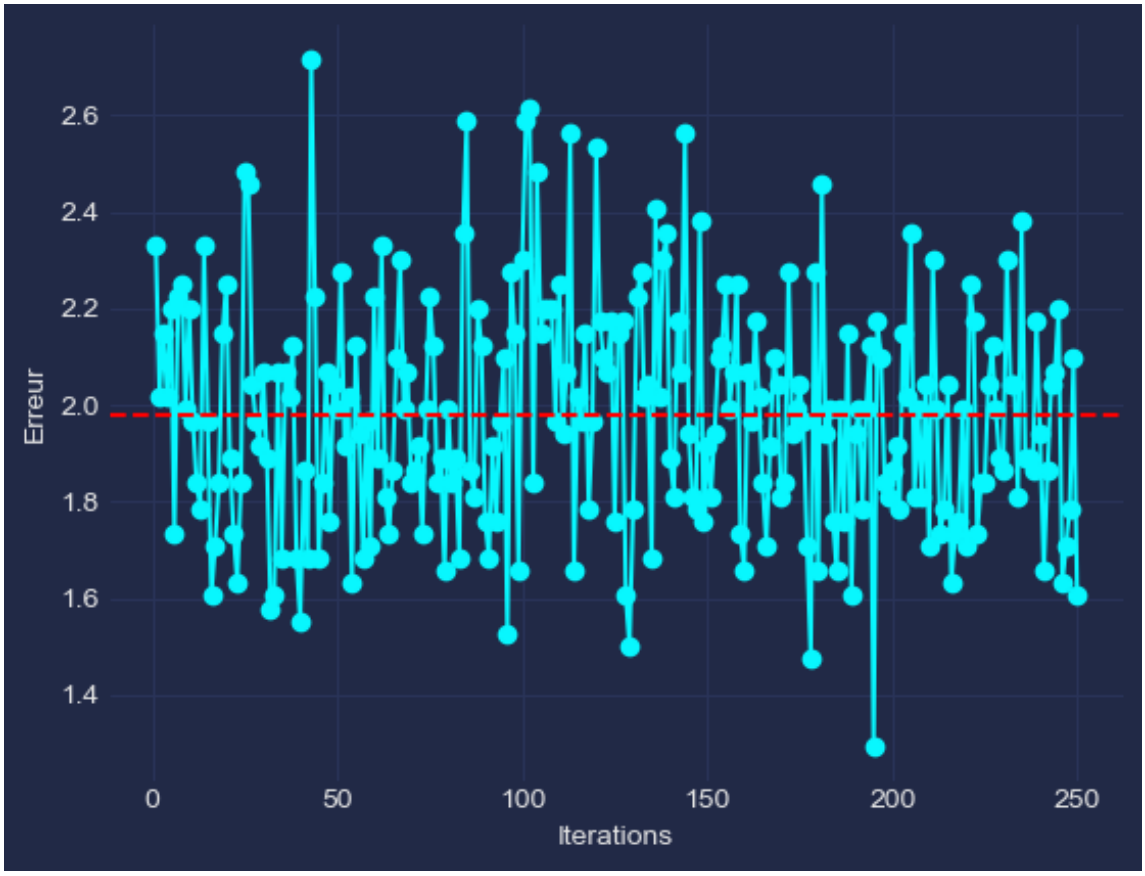

Figure 9 : Evolution de l'erreur du modèle du producteur selon les itérations.

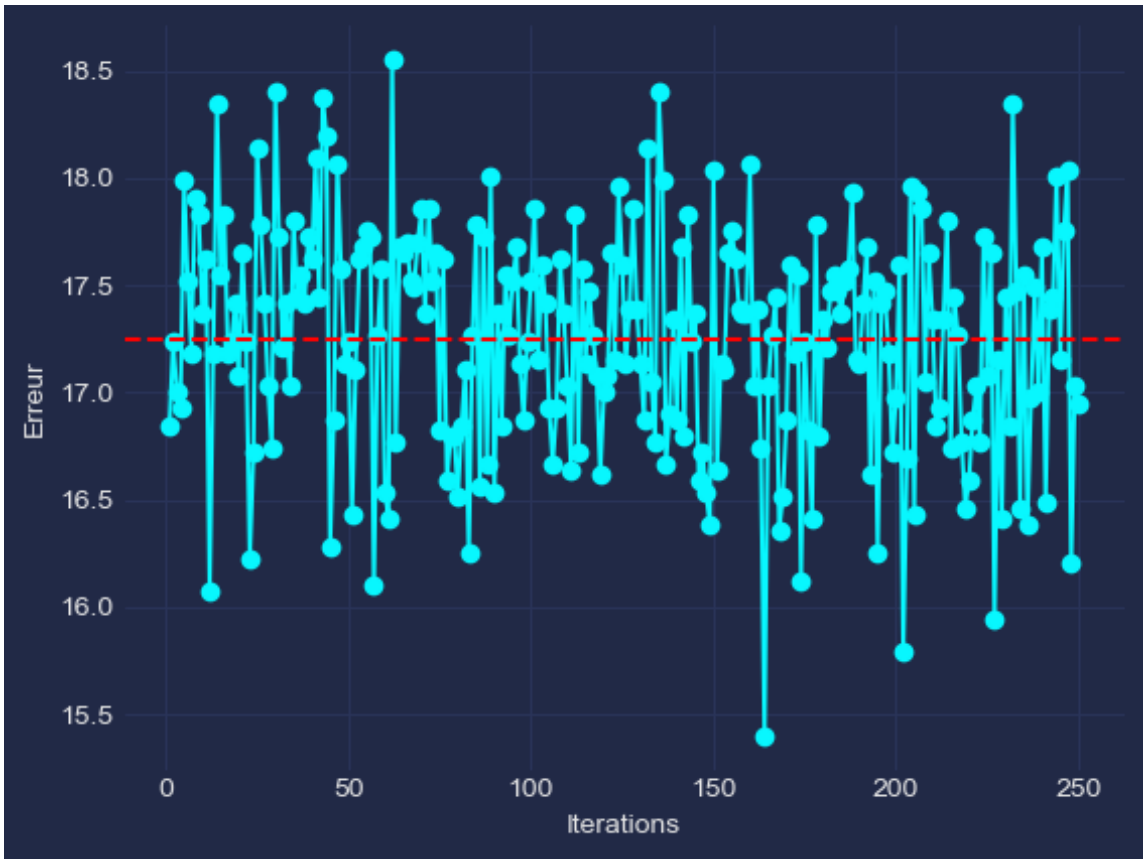

Figure 10 : Evolution de l'erreur du modèle du consommateur selon les itérations.

Nous avons également calculé l'importance de chaque variable (nutriments). Autrement dit, nous avons mesuré l'impact de chaque nutriment sur la précision du modèle en quantifiant la diminution de l'impureté causée par chaque nutriment lors de la construction des arbres de décision. Ainsi, nous avons :

$$Importance_{j,i} := \sum_{t} (p(t) \cdot \Delta I_{j,t,i})$$

Avec $p(t)$ la proportion du nombre d'observations dans le nœud t par rapport au nombre total d'observations dans l'ensemble d'entraînement et $\Delta I_{j,t,i}$ la diminution d'impureté due à la variable j dans le nœud t de l'arbre i. $\Delta I_{j,t,i}$. Nous mesurons $\Delta I_{j,t,i}$ comme la différence entre l'impureté du nœud avant la division et la somme des impuretés des nœuds fils après la division, pondérée par la taille des nœuds fils[1] :

[1] Un nœud fils est un nœud qui résulte de la division d'un nœud parent. Par exemple : un nœud parent rassemble 100 observations, dans le premier nœud fils nous retrouvons un sous-ensemble de 60 observations et le second nœud fils contient l'autre sous-ensemble avec les 40 observations restantes.

$$\Delta I_{j,t,i} := I(t) - \sum_{fils\ de\ t} (\frac{N_f}{N_t} \times I(f))$$

Avec $I(t)$ est l'impureté du nœud t avant la division ; N_f est le nombre d'observations dans le nœud fils f ; N_t le nombre d'observations dans le nœud t et $I(f)$ l'impureté du nœud fils f après la division. Grâce à cela, nous pouvons déterminer l'importance des différents nutriments pour établir le top 3 pour le modèle du producteur et du consommateur (figures 11 et 12). Ainsi, pour le modèle du producteur, les 3 nutriments qui influencent le plus le modèle sont les graisses transformées, les graisses saturées et le calcium. Pour le consommateur, ce sont les graisses saturées, le sel et l'énergie. Grâce à cela, on sait maintenant pourquoi le modèle du consommateur est moins précis que celui du producteur. En effet, la variable la plus importante augmente le plus l'erreur du modèle lorsqu'elle est supprimée. C'est bien ce qu'il se passe ici : les graisses transformées et le calcium absent du modèle du consommateur, leur absence augmente l'erreur du modèle.

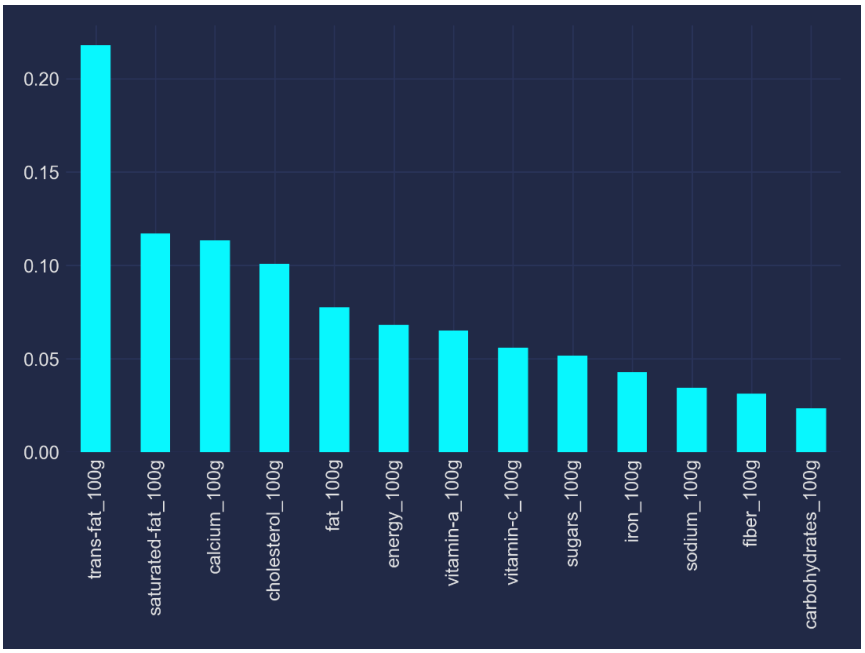

Figure 11 : Importance des variables du modèle du producteur.

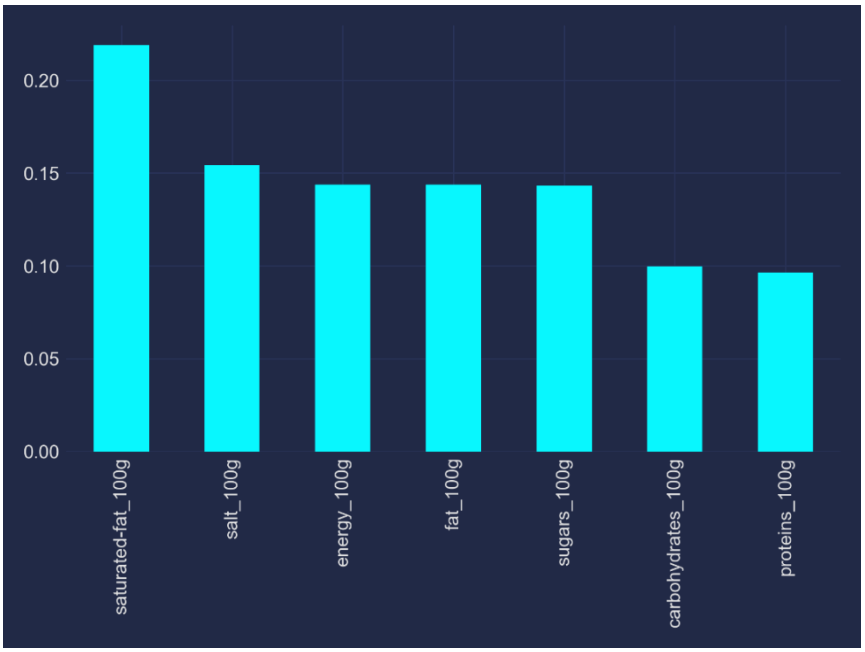

Figure 12 : Importance des variables du modèle du consommateur.

Enfin, lorsque l'on veut faire une prédiction d'un nouveau produit dont les informations seront rentrées dans le formulaire, nous avons besoin d'une fonction de décision notée $f(X)$. Cette fonction est transformée en fonction d'activation softmax (puisque l'on est dans le cas de la classification multi-classe) qui est exprimée ainsi :

$$P(\,i|X) := \frac{e^{f_{i(X)}}}{\sum_{i=1}^{5} e^{f_j(X)}}$$

Pour $i = \{1,2,\ldots,5\}$ étant donné que nous avons 5 classes. La classe prédite est souvent celle ayant la probabilité la plus élevée après l'application de la fonction d'activation : $Prédiction = argmax_i P(classe\ i|X)$.

CONCLUSION

Notre objectif était de prédire le Nutri-Score des produits à partir de leurs valeurs nutritionnelles et que chaque personne puisse évaluer eux même le Nutri-Score de leurs produits. Pour cela nous avons créé un formulaire dans lequel les utilisateurs de notre application puissent rentrer les informations qu'ils ont à leur disposition, selon qu'ils sont consommateur ou producteur. Ces informations sont ensuite stockées dans une variable Excel et sont ensuite utilisées par un algorithme développé sous Python.

Nous avons comparé la régression logistique ordinale et la Random Forest dans le calcul du Nutri-Score, en termes d'erreur de classification. C'est alors que la Random Forest se révélait plus précis que la régression logistique ordinale, car ils ont respectivement une erreur de classification de 2% et 26%, pour le modèle du producteur. Nous avons aussi créé 2 modèles différents utilisant le Random Forest avec les variables du consommateur d'un côté et celles du producteur d'un autre côté. Le deuxième modèle se révèle globalement plus précis que le second, ils ont respectivement une erreur de classification de 2% et 17%. Nous avons conservé les deux modèles sous un pickle pour chacun.

Enfin, Python n'est pas en mesure, à lui seul de récupérer des données directement sur Excel, c'est pour cela que nous avons utilisé les données stocker dans la variable et les pickles enregistrés pour appliquer le modèle. Tout ceci a été possible avec l'utilisation d'une API.

TABLE DES FIGURES

Figure 1 : Page d’accueil. 6

Figure 2 : Formulaire d'entrée. 8

Figure 3 : Dashboard du producteur. 9

Figure 4 : Dashboard du consommateur. 10

Figure 5 : Historique du producteur. 10

Figure 6 : Historique du consommateur. 11

Figure 7 : Matrice de confusion pour le modèle des producteurs. 19

Figure 8 : Matrice de confusion pour le modèle des consommateurs. 20

Figure 9 : Evolution de l’erreur du modèle du producteur selon les itérations. 21

Figure 10 : Evolution de l’erreur du modèle du consommateur selon les itérations. 22

Figure 11 : Importance des variables du modèle du producteur. 23

Figure 12 : Importance des variables du modèle du consommateur. 24

TABLE DES TABLEAUX

Tableau 1 : Description de toutes les variables utilisées. 5

Tableau 2 : Comparaisons des erreurs de classification des différents modèles optimaux..... 17

Tableau 3 : Erreur des modèles après évaluation du Random Forest sur la base de test. 20