

UNIVERSITÉ LYON 1



Extreme Gradient Boosting (XGBoost) et Light-GBM

Poutout Benjamin
Pizzo Gabriel

September 26, 2024

Sommaire

1	Introduction	2
1.1	Motivations	2
1.2	Arbre de Décision	3
1.2.1	Modèle CART	3
1.2.2	Arbre de Classification exemple	3
1.3	Ensemble Learning	5
1.4	Bootstrap Aggregating ou Bagging	7
1.5	Random Forest	8
1.6	Boosting	9
2	Exploration de Données	12
2.1	Exploration de variables et visualisation des données	12
2.1.1	Statistiques descriptives	15
2.2	Recherche et correction d'outliers	22
3	Gradient Boosting de Friedman et XGBoost	30
3.1	XGBoost objectif d'apprentissage généralisé	31
3.2	Algorithme XGBoost	32
3.3	Différences XGB /Gradient Boosting de Friedman	33
3.4	Application : Modèle de Prédiction de Diabète	34
3.4.1	Définition de l'objectif et métriques associées	34
3.4.2	Sélection de variables via Feature Importance	34
3.4.3	Modèles xgboost	38
3.4.4	Evaluation de modèle	42
4	Etude du LightGBM	45
4.1	Algorithme GOSS	45
4.2	Algorithme du Exclusive Feature Bundling	47
4.3	Algorithme LightGBM	50
4.4	Modèle LightGBM	51
4.5	Evaluation de notre modèle LightGBM	52
5	Comparaison avec d'autres modèles	56
5.1	Performances sur des données non structurées	56
5.2	Application aux données non structurées	57
5.3	Conclusion :	60
6	Sources et références	61

1 Introduction

Le Machine Learning est une branche de l'informatique ayant pour but de concevoir des algorithmes permettant d'apprendre à partir de données, on s'intéressera ici à l'apprentissage supervisé. Parmi les algorithmes d'apprentissage, on peut citer les arbres de décisions, la régression linéaire, les k plus proches voisins, réseaux neuronnaux etc... Ces modèles peuvent être appliqués à de la reconnaissance d'image, le traitement du langage naturel ou encore la prédiction de résultats sportifs.

On s'intéressera dans un premier temps aux arbres de classification, qui malheureusement arrivent vite à leur limites et peuvent souffrir de sur ou sous-apprentissage. Ils manquent également d'une généralisation permettant de faire des prédictions précises.

De nouvelles méthodes ont alors été créées, allant de la Forêt Aléatoire qui conceptualise un ensemble d'arbres pour des prédictions plus efficaces à des méthodes comme le XGBoost intégrant également le principe de Boosting d'arbre, ou encore le LightGBM, version améliorée du XGBoost pour le traitement de données de grande dimension. De nombreux domaines utilisent ces technologies, allant du médical avec la détection de maladie à la sécurité et la finance avec la détection de fraude, ou bien même de la reconnaissance vocale.

1.1 Motivations

Dans ce projet proposé par Mr. Cohen, nous allons étudier en détail le XGBoost et le LightGBM, en analysant leur fonctionnement, leur performance et leur capacité à généraliser les données. Le sujet est actuel et nous a tout de suite attiré. Faire des prédictions par de l'apprentissage automatique est challengeant mais extrêmement intéressant.

Nous commencerons tout d'abord par la présentation des bases nous permettant de comprendre le XGBoost, d'où elles viennent et comment les utiliser.

Une grande partie du projet sera dédiée à l'étude du jeu de données sur lequel nous réaliserons des modèles XGBoost et LightGBM. Le pre-processing des données est une partie capitale du travail de tout les Data Scientists, et ce jeu de donnée n'y échappera pas.

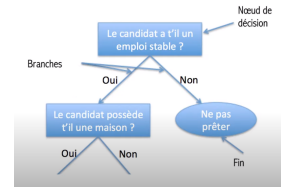
Puis, nous nous intéresserons plus précisément aux algorithmes du XGBoost et du LightGBM, fort de ce que nous aurons appris précédemment.

Ensuite, nous nous focaliserons sur la création, l'étude théorique et l'amélioration de modèles par recherche d'hyperparamètres et réduction de dimension.

Enfin, nous viendrons étudier les performances des différents modèles, que ce soit en termes de performances ou de temps de calcul.

1.2 Arbre de Décision

Un arbre de décision est un algorithme de machine learning qui permet d'effectuer automatiquement des cas de classification et de régression. Il peut résoudre des problèmes linéaires comme non-linéaires, en étant construit comme une **suite de règles** qui permet de résoudre un problème donné.



Un des aspects qui rend l'utilisation d'arbres de décision pertinente (par exemple en classification) est le fait qu'on puisse estimer la probabilité d'appartenance à une classe particulière.

1.2.1 Modèle CART

L'algorithme CART, inventé en 1984 par Breiman, Friedman, Olshen et Stone est un algorithme permettant la création d'un arbre de décision. Concrètement, on commence par regrouper la totalité de nos données dans la racine, représentant le premier nœud.

Ensuite, nous venons créer une division binaire en fonction d'un critère défini au préalable. Nous itérons alors ce processus jusqu'à ce qu'on atteigne le critère d'arrêt également prédéfini.

Dans cet arbre, à chaque nœud terminal sera associé une valeur numérique lorsque la variable à prédire est quantitative, ou une modalité de cette dernière si la variable à prédire est qualitative.

1.2.2 Arbre de Classification exemple

Dans le cas d'une classification de diabète on possède les variables explicatives : "HighBp" qui correspond à la pression artérielle, "GenHealth" qui est un indice global de santé, un arbre de profondeur 2 produira :

Afin de construire les branches d'un arbre de classification il est nécessaire de définir une fonction coût, pour cela on définit un indicateur de "pureté" de chaque nœud qui consiste à mesurer la séparation par nos règles effectuée entre les différentes classes de notre problème parmi la population.

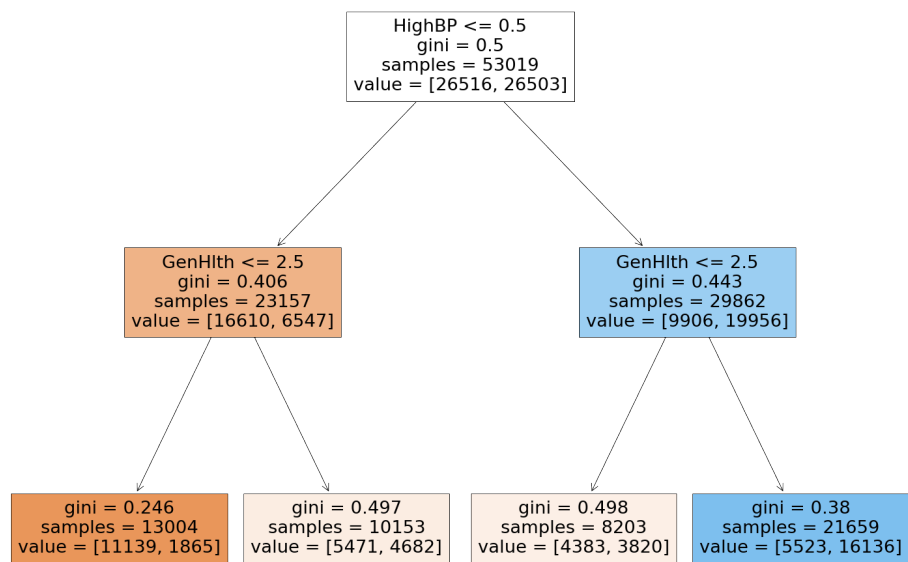


Figure 1: arbre de décision

On définit :

$$G_i = 1 - \sum_{k=1}^n p_{i,k}^2$$

où $p_{i,k}$ est le ratio d'individus de la classe k parmi la population du i ème noeud, plus le noeud est "pur" plus l'indice de gini se rapproche de 0.

Avec l'aide de cet indice on définit alors une fonction coût:

$$J(k) = \frac{m_{gauche}}{M} G_{gauche} + \frac{m_{droite}}{M} G_{droite}$$

Où pour le noeud k , $\frac{m_{gauche}}{M}$ est la proportion du sous ensemble de la population à gauche après séparation pour une règle donnée et G_{gauche} le gini associé, de même pour la sous population de droite.

Il s'agit donc de trouver les règles de séparation minimisant ce coût en testant toutes les valeurs pour chaque variable explicative. Pour des tâches de régression il suffit de remplacer l'indice Gini par la fonction "MSE" qui mesure l'écart à la moyenne au carré.

1.3 Ensemble Learning

Le concept autour de l'Ensemble Learning se base sur une idée particulière. En effet, on suppose que la perception et la résolution d'un problème est plus efficace par une foule que par n'importe quel individu. Il faut en revanche satisfaire trois hypothèses :

Tout d'abord, la **diversité** : avoir des personnes de divers milieu avec des idées originales, puis vient **l'indépendance** : permettre à ces divers avis de s'exprimer sans aucune influence, et enfin **la décentralisation** : additionner ces différents jugements plutôt que de laisser une autorité supérieure choisir les idées qu'elle préfère.

Cette idée sera par la suite adaptée mathématiquement et le concept est très pratique en machine learning.

Les méthodes d'Ensemble Learning utilisent plusieurs algorithmes d'apprentissage et prennent en compte les résultats de ces modèles afin d'obtenir de meilleures performances prédictives que les modèles pris séparément.

Bien entendu, il faudra des modèles bien **différents**, construits sur des données ou des algorithmes d'apprentissage différents.

Index	Modèle 1	Modèle 2	Modèle 3	Modèle 4	Modèle 5	Mélange
1	1	1	0	0	1	1
2	1	1	0	1	0	1
3	0	0	1	1	1	1
4	0	1	1	1	0	1
5	1	0	1	0	1	1
Perf	60%	60%	60%	60%	60%	100%

Figure 2: Vote Majoritaire

Ce concept sera la base pour les **randoms forests** (Bagging/Pasting), le **boosting** (ADABOOST, Gradient Boosting) ainsi que le **stacking** (utilisation d'un modèle afin de mélanger les prédictions des modèles).

1.4 Bootstrap Aggregating ou Bagging

Le Bootstrap Aggregating, plus communément appelé Bagging est un méta-algorithme d'apprentissage ensembliste conçu pour améliorer la stabilité et la précision de modèles en apprentissage automatique. Il consiste en la réduction de variance en utilisant différents sous-ensembles d'entraînement, puis en combinant les différentes prédictions individuelles pour produire une prédiction globale plus stable et précise.

Soit $X = (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ un ensemble de données d'entraînement

où chaque x_i est un vecteur d'une variable explicative et y_i est la target. Nous rappelons que nous voulons prédire y .

Le processus de Bagging fonctionne comme suit :

1. **Selection** des sous-ensembles de données à partir de notre jeu de données :

Simplement, nous venons effectuer un échantillonnage avec remplacement sur l'ensemble de données X pour créer un nouveau sous-ensemble de données X_i . Notons que chaque observation X_i peut être sélectionnée de multiples fois.

Soit $X_i = (x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}), \dots, (x_{i_m}, y_{i_m})$ le sous-ensemble de données aléatoirement sélectionné à partir de X .

où $m \leq n$ représente la taille du notre nouveau sous-ensemble de données.

2. **Utilisation** de chaque sous-ensemble de données créé pour entraîner tous nos modèles.

Nous allons entraîner un modèle individuel sur chaque sous-ensemble de données X_i . Nous pouvons utiliser n'importe quel algorithme d'apprentissage automatique pour entraîner chaque modèle individuel. Par exemple, si nous utilisons des arbres de décision, nous allons entraîner un arbre de décision sur chaque sous-ensemble de données X_i .

Soit $f_i(x)$ le modèle individuel entraîné sur le sous-ensemble de données X_i .

3. **Combinaison** des prédictions de nos nouveaux modèles créés pour produire une prédiction globale.

Nous regardons la prédiction moyenne de chaque modèle. Comme nous nous retrouvons dans un cas de classification binaire, pour combiner les prédictions on utilise l'ensemble learning *ie* un vote majoritaire.

On a alors :

$$F(x) = \frac{1}{M} \sum_{i=1}^M f_i(x)$$

qui représente la prédiction finale, où M est le nombre de modèles individuels entraînés.

1.5 Random Forest

Lorsqu'on utilise un arbre seul, peu importe sa profondeur ou sa complexité, on s'aperçoit de son manque de flexibilité pour donner de bonnes prédictions sur des données de test hors du jeu de base. L'arbre créé peu être très juste sur ses données, mais généralise très mal (overfitting). Or, le but d'un algorithme de machine learning est de pouvoir classer des données dont on ne connaît pas encore la prédiction.

Pour palier à ce problème, on vient utiliser plusieurs arbres de décision afin de créer une forêt et d'améliorer la généralisation de l'ensemble du modèle.

Construction d'une Random Forest:

1. **Sélection** donnée par donnée aléatoirement notre jeu de données (on se place dans la méthode de Bagging donc on peut bien entendu prendre plusieurs fois la même observation dans notre jeu pour l'arbre que l'on va créer). Il est capital de **maximiser la variété des arbres**. Cela permet une compensation mutuelle des arbres par rapport à leur forces et faiblesses relatives.
2. **Entraînement de l'arbre** à partir du jeu de données qu'on vient sélectionner aléatoirement. On va sélectionner aléatoirement une partie des variables pour entraîner notre arbre.
3. **Répétition du processus** jusqu'à avoir créé autant d'arbres qu'on le souhaite.

Comment l'algorithme du Random Forest calcule ses prédictions ? :

Soit alors :

$$y_{i,t} = f_t(x_i)$$

nous avons $\hat{y}_{i,t}$ comme prédiction de l'arbre t pour l'observation i , et f_t est notre fonction de prédiction de l'arbre t et x_i est l'observation i .

En prenant la moyenne des prédictions de tous les arbres nous obtenons :

$$\hat{y}_i = \frac{1}{T} \sum_{t=1}^T y_{i,t}$$

où \hat{y}_i représente la prédiction finale pour l'observation i et T est le nombre total d'arbres de décision dans la forêt aléatoire.

1.6 Boosting

Le boosting est une méthode d'ensemble learning qui consiste à combiner plusieurs modèles faibles en un modèle fort. Cette méthode utilise une approche itérative, où chaque modèle est entraîné sur un sous-ensemble différent des données d'entraînement, amélioré selon les erreurs de prédiction du modèle précédent.

Plus précisément, soit un ensemble d'apprentissage :

$$\mathcal{S} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

où x_i représente le i -ème exemple et y_i sa variable cible. Soit également $h_t(x)$ le modèle faible appris à l'itération t , et $w_{i,t}$ le poids du i -ème exemple à l'itération t . Initialement, tous les poids sont égaux à $1/n$.

À chaque itération t , un nouveau modèle faible $h_t(x)$ est appris sur les exemples $S_t = (x_i, y_i, w_{i,t})_{i=1\dots n}$, où $w_{i,t}$ représente le poids du i -ème exemple à l'itération t . Ces poids sont mis à jour à chaque itération en fonction des erreurs de prédiction du modèle précédent $h_{t-1}(x)$. On a alors :

$$w_{i,t} = \frac{w_{i,t-1} e^{-\alpha y_i h_{t-1}(x_i)}}{Z_t}$$

où y_i est l'étiquette de l'exemple d'entraînement i , $h_{t-1}(x_i)$ est la prédiction actuelle du modèle combiné sur l'exemple i à l'itération $t-1$, α_t est le coefficient d'apprentissage de l'itération t et Z_t est une constante de normalisation appelée facteur de normalisation de Boltzmann :

$$Z_t = \sum_{j=1}^n w_{j,t} e^{-y_j h_{t-1}(x_j)}$$

Une fois que tous les modèles faibles ont été entraînés, le modèle fort final $H(x)$ est défini comme une somme des modèles faibles :

$$H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$$

où T est le nombre total d'itérations.

L'objectif de la méthode de boosting est de minimiser la somme des erreurs de prédiction sur les exemples d'entraînement S :

$$\epsilon_t = \text{sign}\left(\sum_{i=1}^n w_{i,t} [y_i \neq h_t(x_i)]\right)$$

où $[y_i \neq h_t(x_i)]$ est une fonction indicatrice qui vaut 1 si la prédiction est incorrecte et 0 sinon.

Ainsi, la fonction de perte optimisée pour l'itération t est donnée par :

$$L_t = \sum_{i=1}^n w_{i,t} e^{-\alpha_t y_i h_t(x_i)}$$

Le coefficient d'apprentissage α_t est choisi de manière à minimiser cette fonction de perte :

$$\alpha_t = \frac{1}{2} \ln \left(\frac{\epsilon_t}{1 - \epsilon_t} \right)$$

où ϵ_t est l'erreur de prédiction du classificateur faible t .

Algorithmiquement parlons, nous aurons alors la méthode suivante :

Entrée: Séquence de N exemples labélisés $(x_1, y_1), \dots, (x_N, y_N)$, distribution D sur les N exemples, un algorithme *WeakLearn*, un entier T qui spécifie le nombre d'itérations.

1. **Initialiser** le vecteur de poids : $w_{1,i} = D(i)$ pour $i = 1, \dots, N$.
2. **Do pour** $t = 1, 2, \dots, T$
 - (a) Initialiser $p_t = \frac{\sum_{i=1}^N w_{t,i}}{N}$.
 - (b) Appeler *WeakLearn*, nous lui donnons la distribution p_t ; et nous obtenons l'hypothèse $h_t : X \rightarrow 0, 1$.
 - (c) Calculer l'erreur de h_t : $\epsilon_t = \sum_{i=1}^N p_{t,i} |h_t(x_i) - y_i|$.
 - (d) Initialiser $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$.
 - (e) Initialiser les nouveaux vecteurs de poids pour être $w_{t+1,i} = w_{t,i} e^{-\alpha_t y_i h_t(x_i)}$.
3. **Renvoie** l'hypothèse $h_f(x) = \begin{cases} 1 & \text{si } \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t, \\ 0 & \text{sinon.} \end{cases}$

2 Exploration de Données

2.1 Exploration de variables et visualisation des données

Afin d'établir un modèle il est essentiel d'analyser nos variables explicatives afin d'établir un lien entre la cible et les variables explicatives.

Pour cela nous pourrions étudier les statistiques descriptives des variables observées, établir l'importance des variables via l'indice gini, les tests d'associations. Dans notre exemple la variable cible sera la positivité au Diabète.

Les variables détaillées sont :

- Age : 13 catégories d'âge, avec : 1 = 18-24 ans / 2 = 25-29 ans / 3 = 30-34 ans / 4 = 35-39 ans / 5 = 40-44 ans / 6 = 45-49 ans / 7 = 50-54 ans / 8 = 55-59 ans / 9 = 60-64 ans / 10 = 65-69 ans / 11 = 70-74 ans / 12 = 75-79 ans / 13 = 80 ans ou plus
- Sex : Genre du patient (1: Homme; 0: Femme)
- HighChol: 0 = Taux de cholestérol faible, 1 = Taux de cholestérol fort
- CholCheck: 0 = Pas de test pour le cholestérol en 5 ans, 1 = Il y a eu un test pour le cholestérol dans les 5 ans.
- BMI: IMC (Indice Masse Corporelle)
- Smoker: Avez vous fumé plus de 100 cigarettes dans votre vie ? 0 = Non, 1 = Oui
- HeartDiseaseorAttack: Maladie Cardio Vasculaire (CHD) or Infection Myocardiale (MI) 0 = Non, 1 = Oui
- PhysActivity: Activité physique durant les 30 derniers jours -n'incluant pas le travail 0 = Non, 1 = Oui
- Fruits: Manger plus d'un fruit par jour 0 = Non, 1 = Oui
- Veggies: Manger au moins un légume par jour = Non, 1 = Oui
- HvyAlcoholConsump: 14 verres par jour pour un homme adulte, et 7 verres par jour pour une femme adulte 0 = Non, 1 = Oui
- GenHlth: Comment noteriez vous votre santé générale : Echelle 1-5 1 = Excellent, 2 = Très Bonne , 3 = Bonne, 4 = Décente, 5 = Faible
- MentHlth: Nombre de jours ou on se sent moins bien mentalement par mois 1-30 jours
- PhysHlth: Blessure physique dans les 30 derniers jours Echelle : 1-30

- DiffWalk: Avez vous des difficultés pour monter les escaliers ? 0 = Non 1 = Oui
- Stroke: Avez vous déjà eu un arrêt cardiaque 0 = Non, 1 = Oui
- HighBP: 0 = Faible BP , BP 1 = fort BP
- Diabetes: 0 = Pas de Diabète, 1 = Diabète

Maintenant que le jeu de données nous paraît un peu plus compréhensible, nous allons effectuer une analyse approfondie de notre dataset, pour vérifier (et modifier si besoin) que notre jeu intégrale soit propre, et qu'on puisse faire tourner nos modèles sur celui-ci sans incohérences.

Il y a plusieurs choses à prendre en considération :

Le type de nos données qui devra être cohérent à l'emploi d'un classifieur XGBoost, nous allons donc vérifier que tout correspond :

```

Age                float64
Sex                float64
HighChol           float64
CholCheck          float64
BMI                float64
Smoker             float64
HeartDiseaseorAttack float64
PhysActivity       float64
Fruits             float64
Veggies           float64
HvyAlcoholConsump float64
GenHlth            float64
MentHlth           float64
PhysHlth           float64
DiffWalk           float64
Stroke             float64
HighBP             float64
Diabetes           float64
dtype: object

```

Figure 3: types données

Les types affichés ci-dessus correspondent bien à ce que l'on voudrait pour pouvoir utiliser le XGoost.

Regardons maintenant si il y a des valeurs manquantes dans notre dataset. Si données manquantes il y a, on utilise une technique qui associe à la case vide une valeur qui est par exemple la moyenne des valeurs de la colonne ou la case est manquante.

```

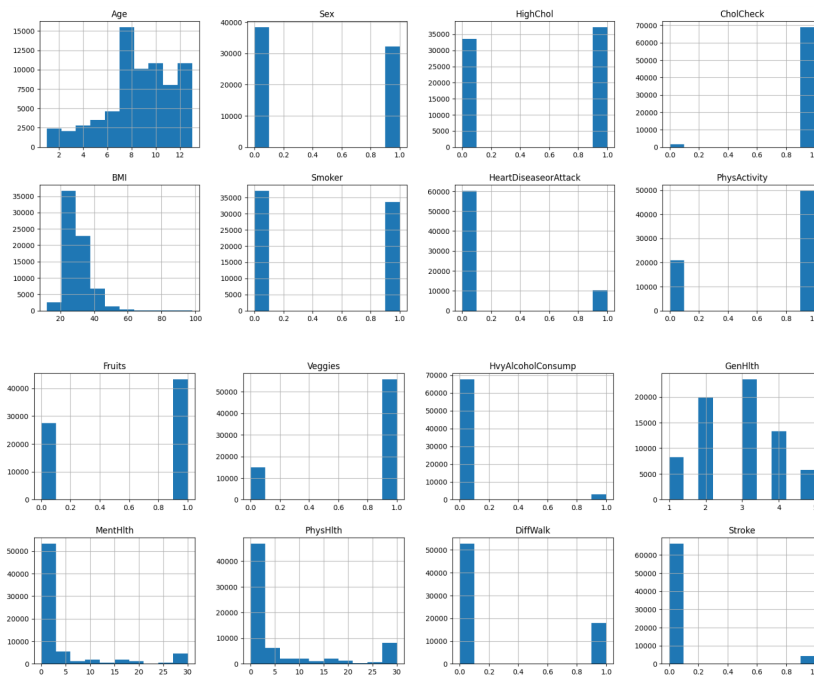
Age          0
Sex          0
HighChol     0
CholCheck    0
BMI          0
Smoker       0
HeartDiseaseorAttack 0
PhysActivity 0
Fruits       0
Veggies     0
HvyAlcoholConsump 0
GenHlth     0
MentHlth    0
PhysHlth    0
DiffWalk    0
Stroke      0
HighBP      0
Diabetes    0
dtype: int64

```

Figure 4: valeurs manquantes

Il n'y a ici pas de valeurs manquantes, on va pouvoir maintenant visualiser et explorer plus profondément nos données.

Pour commencer, un histogramme de nos variables pour voir à quoi ressemblerait la distribution dans chaque classe :



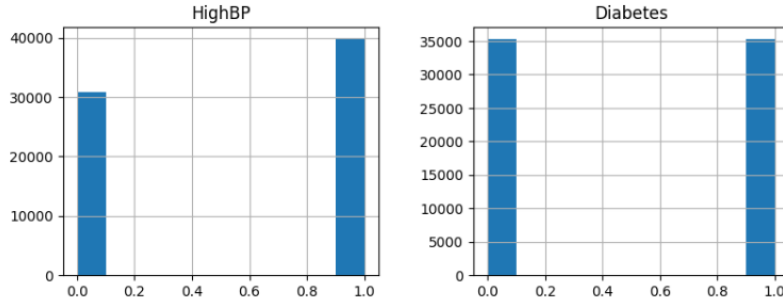


Figure 5: Distribution des données par histogramme

Dans cet histogramme, nous observons plusieurs choses :

Premièrement, nous pouvons constater que le jeu de données est équilibré car il y a autant d'observation de patients diabétique que de non diabétiques. Il n'y aura donc pas de rééquilibrage à effectuer.

Notons que si les données ne sont pas équilibrées, il faut le faire soit en réduisant son dataset pour garder autant de cas possibles soit en spécifiant que l'on choisi le même ratio dans l'échantillonnage de notre jeu.

2.1.1 Statistiques descriptives

Pour une variable x avec n observations, on utilise couramment la moyenne, l'écart-type, la médiane, le minimum et le maximum pour faire une analyse statistique descriptive.

La moyenne est calculée en additionnant toutes les observations et en divisant par le nombre d'observations n :

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

L'écart-type est une mesure de la dispersion des données autour de la moyenne, et est calculé en soustrayant chaque observation de la moyenne, en élevant au carré et en faisant la moyenne de ces différences, puis en prenant la racine carrée de ce résultat :

$$\sigma = \sqrt{\frac{1}{n} \sum (x_i - \bar{x}_i)^2}$$

	Age	Sex	HighChol	CholCheck	BMI	Smoker	HeartDiseaseorAttack	PhysActivity	Fruits
count	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000
mean	8.584055	0.456997	0.525703	0.975259	29.856985	0.475273	0.147810	0.703036	0.611795
std	2.852153	0.498151	0.499342	0.155336	7.113954	0.499392	0.354914	0.456924	0.487345
min	1.000000	0.000000	0.000000	0.000000	12.000000	0.000000	0.000000	0.000000	0.000000
25%	7.000000	0.000000	0.000000	1.000000	25.000000	0.000000	0.000000	0.000000	0.000000
50%	9.000000	0.000000	1.000000	1.000000	29.000000	0.000000	0.000000	1.000000	1.000000
75%	11.000000	1.000000	1.000000	1.000000	33.000000	1.000000	0.000000	1.000000	1.000000
max	13.000000	1.000000	1.000000	1.000000	98.000000	1.000000	1.000000	1.000000	1.000000

Figure 6: Statistiques descriptives pour chaque variable explicative

	Veggies	HvyAlcoholConsump	GenHlth	MentHlth	PhysHlth	DiffWalk	Stroke	HighBP	Diabetes
70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000
0.788774	0.042721	2.837082	3.752037	5.810417	0.252730	0.062171	0.563458	0.500000	
0.408181	0.202228	1.113565	8.155627	10.062261	0.434581	0.241468	0.495960	0.500004	
0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1.000000	0.000000	3.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.500000	
1.000000	0.000000	4.000000	2.000000	6.000000	1.000000	0.000000	1.000000	1.000000	
1.000000	1.000000	5.000000	30.000000	30.000000	1.000000	1.000000	1.000000	1.000000	

Figure 7: Statistiques descriptives pour chaque variable explicative

Nous pouvons aller plus loin en utilisant un graphique bivarié pour essayer de comprendre encore plus la relation entre deux variables. Le type de graphique dépendra du niveau de mesure des variables (catégorielle ou quantitative).

Pour les variables catégorielles, nous aurons alors :

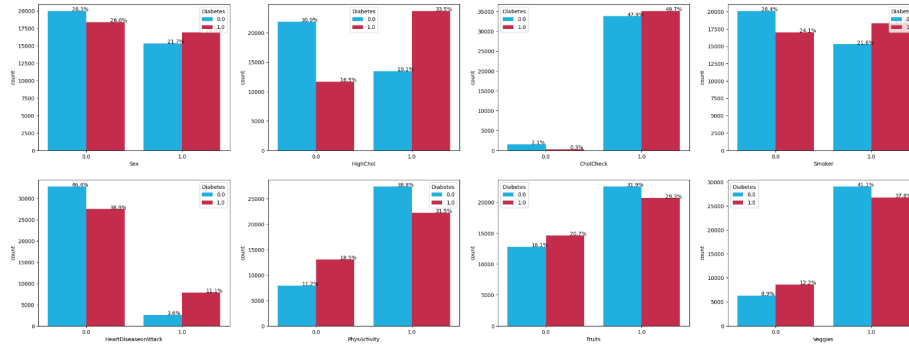


Figure 8: Proportion diabète par variable catégorielle

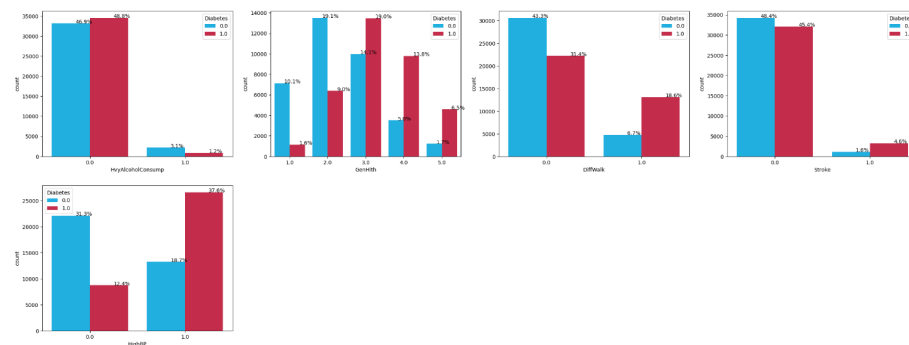


Figure 9: Proportion de Diabète par variable catégorielle

Egalement, nous pouvons représenter les déviations de moyennes de taux moyens de diabète en fonction de la valeur prise par une variable catégorielle : (valeur moins moyenne globale)

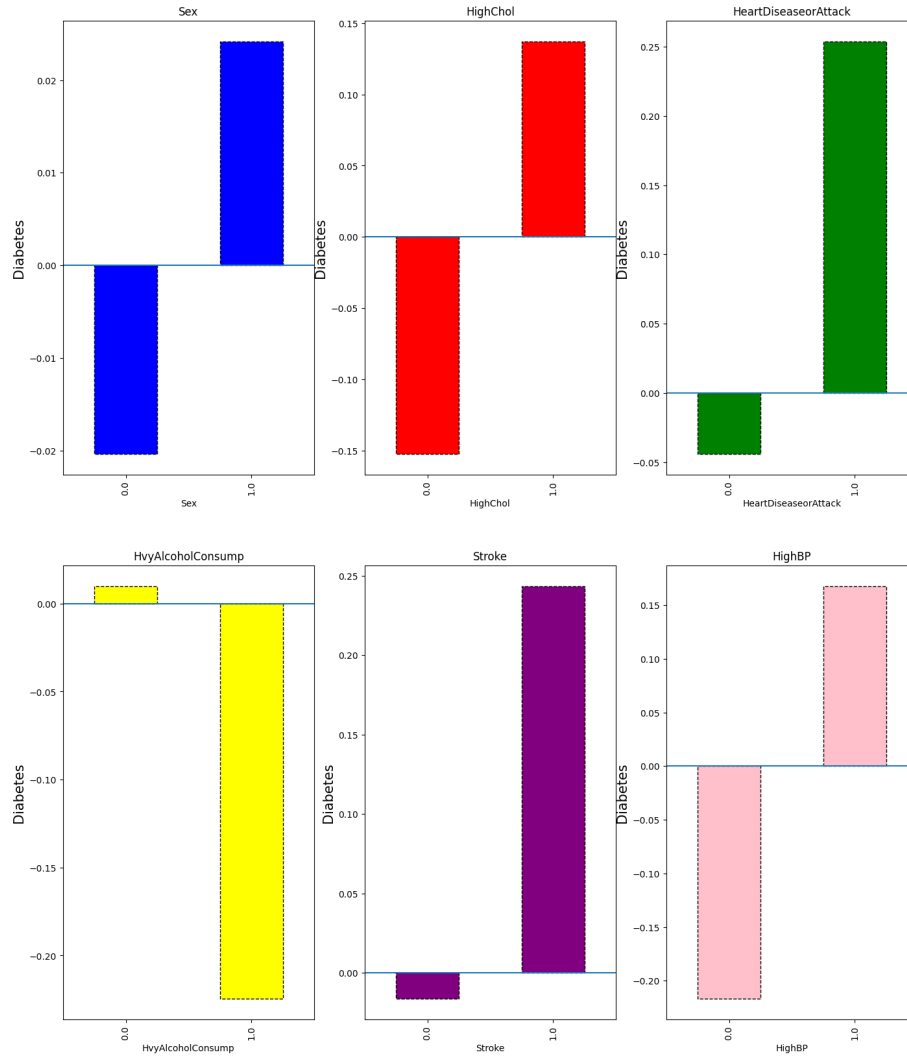


Figure 10: Déviations De Moyennes

En vérifiant l'échelle on voit que par exemple le genre a un effet peu significatif mais que la consommation d'alcool si.

Tests ANOVA

Analyse de la variance (ANOVA) est une formule statistique utilisée pour comparer les variances entre la ou les moyennes de différents groupes. Elle est utilisée dans de nombreux scénarios pour déterminer s'il existe une différence entre les moyennes de différents groupes.


	feature	f_value	p_value	
0	Sex	1.397117e+02	3.300778e-32	
1	HighChol	6.452512e+03	0.000000e+00	
2	CholCheck	9.537878e+02	4.864796e-208	
3	Smoker	5.267061e+02	3.918885e-116	
4	HeartDiseaseorAttack	3.310962e+03	0.000000e+00	
5	PhysActivity	1.825563e+03	0.000000e+00	
6	Fruits	2.073232e+02	6.143416e-47	
7	Veggies	4.472687e+02	5.752654e-99	
8	HvyAlcoholConsump	6.417805e+02	5.854501e-141	
9	DiffWalk	5.676790e+03	0.000000e+00	
10	Stroke	1.129861e+03	9.382035e-246	
11	HighBP	1.204198e+04	0.000000e+00	
12	Diabetes	inf	0.000000e+00	

Figure 11: Tests ANOVA

Hypothèse nulle (H0) : Elle se produit lorsqu'il n'y a pas de différence entre les groupes ou les moyennes. En fonction du résultat du test ANOVA, l'hypothèse nulle sera soit acceptée, soit rejetée.

Hypothèse alternative (H1) : Lorsqu'une théorie veut qu'il existe une différence entre les groupes et les moyennes.

Le résultat d'ANOVA est la "statistique F". Ce ratio montre la différence entre la variance à l'intérieur du groupe et la variance entre les groupes, ce qui produit finalement un chiffre qui permet de conclure que l'hypothèse nulle est

soutenue ou rejetée. S'il existe une différence significative entre les groupes, l'hypothèse nulle n'est pas soutenue, et le ratio F sera plus grand.

Cela se traduit mathématiquement de la manière suivante :

1. Nous commençons par choisir notre seuil α en général nous prendrons 0.05.
2. Puis nous collectons les données dans chacun des groupes afin de calculer leur moyennes $\bar{x}_1, \bar{x}_2 \dots \bar{x}_n$.
3. Ensuite, on calcule la Variance Totale obtenue :

$$TotalVar = \sum_{i=1}^k \sum_{j=1}^{n_i} (x_{ij} - \bar{x})^2$$

où \bar{x} est la moyenne de toutes les observations :

$$\bar{x} = \frac{\sum_{i=1}^k \sum_{j=1}^{n_i} x_{ij}}{n}$$

ou n est le nombre d'observations total et n_i le nombre d'observation par groupe.

4. On calcule la somme des carrés de chaque groupe qui nous servira pour la formule finale de la f-statistique :

$$SG = \sum_{i=1}^k n_i (\bar{x}_i - \bar{x})^2$$

5. On calcule également la somme des carrés des résidus totaux :

$$SR = \sum_{i=1}^k \sum_{j=1}^{n_i} (x_{ij} - \bar{x}_i)^2$$

6. On initialise les degrés de liberté des groupes et des résidus tel que pour les groupes on ait $DG = k - 1$ et $DR = n - k$.

7. On calcule finalement la f-statistique :

$$F = \frac{\frac{SG}{DG}}{\frac{SR}{DR}}$$

Nous pouvons aussi voir la relation de variables quantitatives par des graphes séparant la distribution des données en fonction de si la variable cible (Diabète) vaut 1 ou 0.

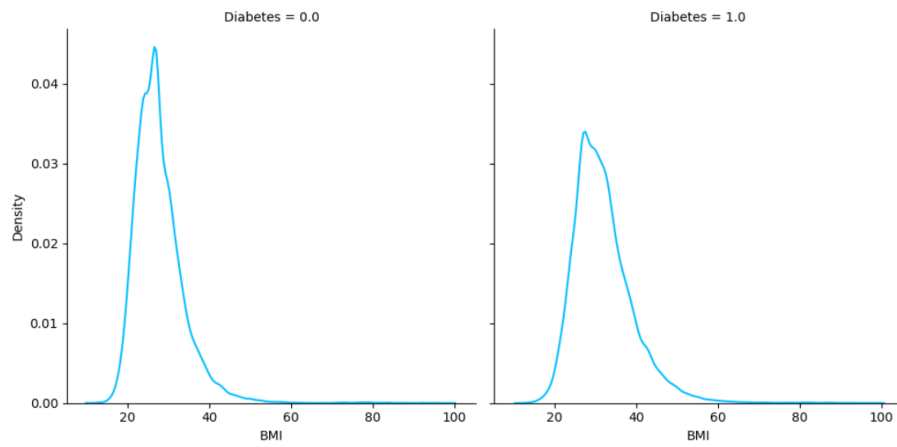


Figure 12: Distribution par Valeur de Diabète

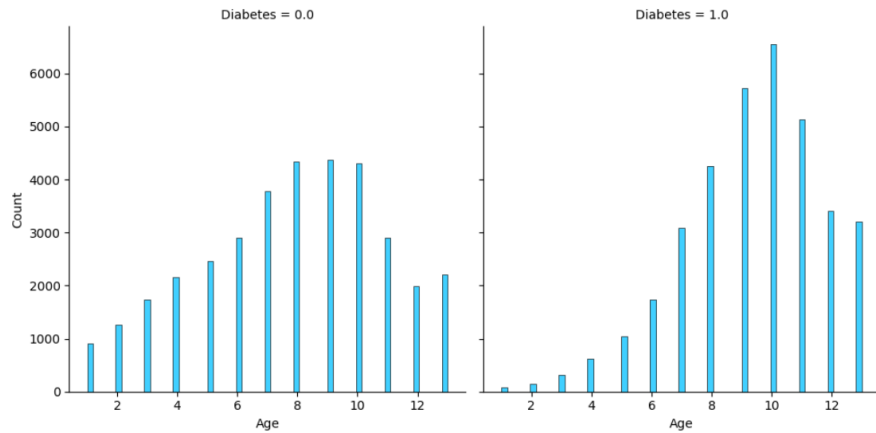


Figure 13: Distribution par Valeur de Diabète

Nou constatons bien une distribution différente pour ces variables quantitatives.

2.2 Recherche et correction d'outliers

Une part importante du pre-processing des données est la recherche et le traitement de valeurs abberantes, dites "outliers".

En effet, des valeurs abberantes présentent dans le jeu de données peuvent avoir un impact non négligeable sur les résultats de nos modèles, car ces valeurs peuvent fausser nos statistiques et modèles prédictifs.

Il sera donc important d'explorer et de traiter ces outliers correctement, afin d'obtenir de meilleurs prédictions.

La présence d'outliers s'explique en diverses raisons. Des erreurs de mesure, des erreurs de saisie ou des événements rares et inattendus peuvent être la cause de ces événements.

Pour détecter les outliers, plusieurs méthodes existent. Pour notre part, nous allons utiliser des diagrammes de boîtes et de moustaches, qui permettent de visualiser les valeurs aberrantes à l'aide de graphiques.

Ci dessous, nous représentons donc chaque variable explicative par son diagramme de boîte à moustache :

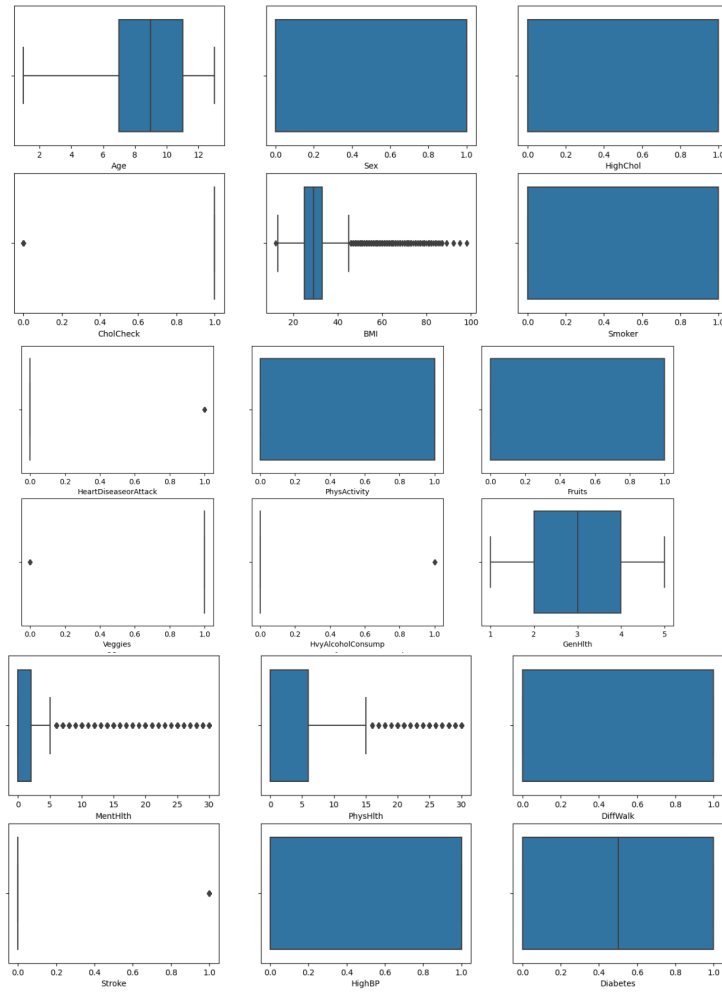


Figure 14: Boîtes à moustaches

Nous remarquons alors ici bien une présence d'outliers dans certaines variables explicatives de notre dataset.

Comment trouver ces outliers ? Pour se faire, on commence par trouver les quantiles de chaque variables explicatives. Pour chacune d'entre elle, nous allons calculer l'IQR (Inter Quartile Range):

$$IQR = Q_3 - Q_1$$

Puis, on calcule les frontières limites qui vont délimiter les données en donnée "normale" ou "abberante". Cette frontière s'exprime :

$$\mathcal{F}_{haute} = Q_3 + 1.5 * IQR$$

$$\mathcal{F}_{basse} = Q_1 - 1.5 * IQR$$

Avant toute manipulation sur les données, il faut rester vigilant. Certaines de nos variables explicatives pourraient disparaître avec le traitement des valeurs abberantes *ie* des variables catégorielles (0 ou 1) pourraient se voir transformer ou en 0 ou en 1 et ne laisser du coup qu'une variable explicative contenant une seule valeur, une variable donc qu'on pourrait hypothétiquement retirer de notre dataset.

Dans le cas ou nous nous rendrions compte par les études des différents tests qui existent que certaines variables explicatives ne sont pas ou très peu utilisées par notre modèle, alors on pourra venir d'abord remplacer les outliers et supprimer ces variables explicatives supplémentaires.

Matrice de corrélation :

Dans un premier lieu nous pouvons visualiser les associations linéaires entre variables, pour celà nous allons afficher la matrice de corrélation avec de coefficient de Pearson, à noter : certaines de nos variables sont catégoriques cette métrique n'est donc pas adaptée pour rendre compte de toutes les associations de variables.

Pour étudier l'importance des données et ainsi savoir si il est possible ou non de se débarrasser de certaines variables *ie* effectuer de la réduction de dimension.

Nous avons alors pour matrice de corrélation :

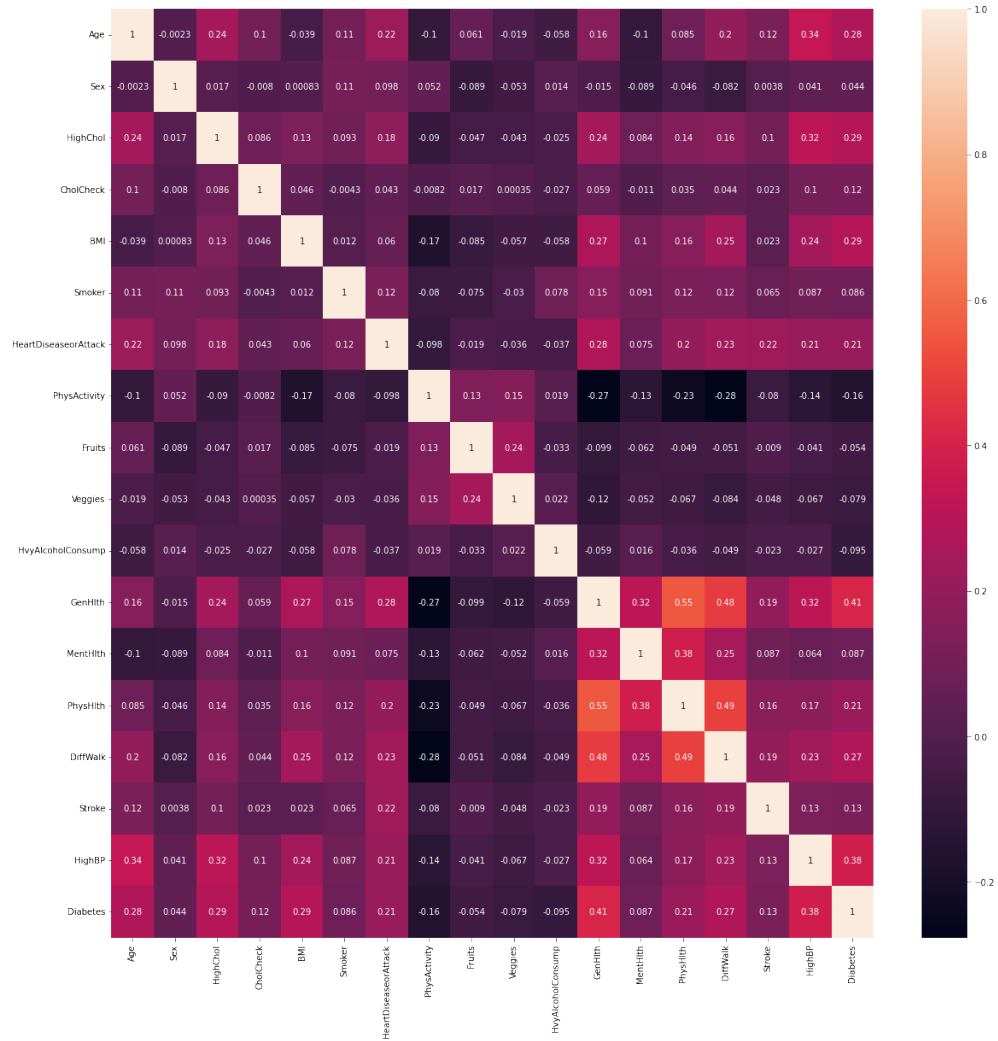


Figure 15: Matrice de Correlation

Les coefficients de la matrice sont définis par :

$$M_{i,j} = \frac{Cov(X_i, X_j)}{\sigma_{X_i} \sigma_{X_j}}$$

où : $Cov(X, Y)$ et la covariance des variables X et Y, σ_X l'écart-type de la variable X. On peut écrire la formule explicite comme :

$$M_{i,j} = \frac{\sum(X_i - \bar{X}_i) \sum(X_j - \bar{X}_j)}{\sqrt{\sum(X_i - \bar{X}_i)^2} \sqrt{\sum(X_j - \bar{X}_j)^2}}$$

Analyse en composantes principales :

L'analyse en composantes principales est une méthode de réduction de dimension, elle est basée sur des projections linéaires et est donc utile lorsque les variables sont entre-corrélées donc quand les coefficients de la matrice de corrélations sont élevés.

Soit un ensemble de données contenant N observations (lignes) et p variables (colonnes). Nous venons alors représenter ces toutes ces données dans une matrice X de taille $N \times p$, où chaque ligne représente une observation et chaque colonne représente une variable. Après avoir calculé la moyenne de chaque variable explicative, nous soustrayons cette moyenne de chaque observation, de sorte que la matrice X soit centrée en zéro.

Le but du PCA est de trouver un ensemble de nouvelles variables (les composantes principales) qui soient des combinaisons linéaires des variables originales, de sorte que les nouvelles variables soient orthogonales les unes aux autres et triées par ordre décroissant d'importance. Pour résumer, la première composante principale devrait expliquer le plus possible de la variance totale des données, la deuxième composante principale devrait expliquer le plus possible de la variance restante, et ainsi de suite.

Concrètement, nous utilisons la décomposition en valeur propres de la matrice de covariance C de X . La matrice de covariance est définie comme suit :

$$C = \frac{1}{N-1} X^T X \quad (1)$$

où X^T représente la transposée de X . La matrice de covariance mesure la variance de chaque paire de variables dans la matrice X .

Ensuite, nous pouvons calculer les vecteurs propres de la matrice de covariance C .

Enfin, en multipliant la matrice centrée en zéro X par les vecteurs propres triés de C , nous obtenons nos composantes principales.

La première composante principale sera donnée par :

$$V_1 = XW_1 \quad (2)$$

où W_1 est le vecteur propre correspondant à la plus grande valeur propre de C , et V_1 est la première composante principale.

Passage de 2 à 1 dimension

En disposant de deux variables corréllées entre elles, on peut faire passer une droite de régression minimisant la distance au sens des moindre carrés, on peut projeter sur cette axe les points :

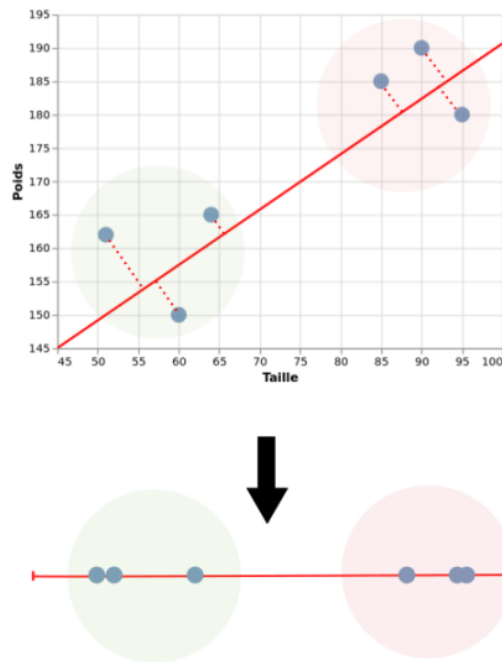


Figure 16: Projection sur la Composante Principale

Cette projection permet de réduire de passer de deux à une dimension, mais en perdant de l'information, c'est pourquoi il est important de garder la quantité d'information conservée.

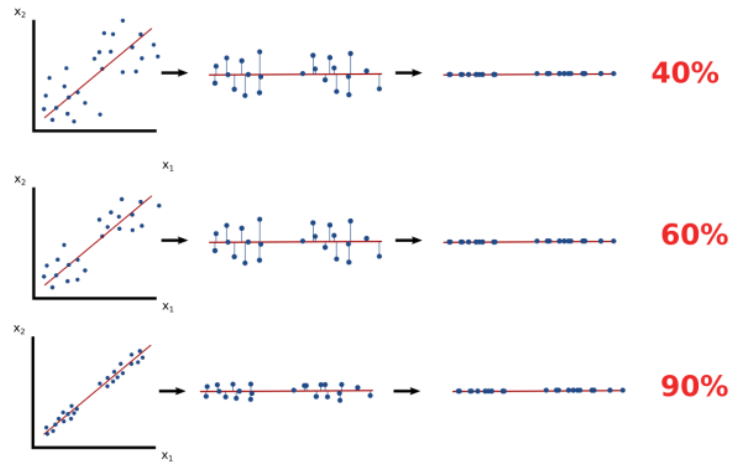


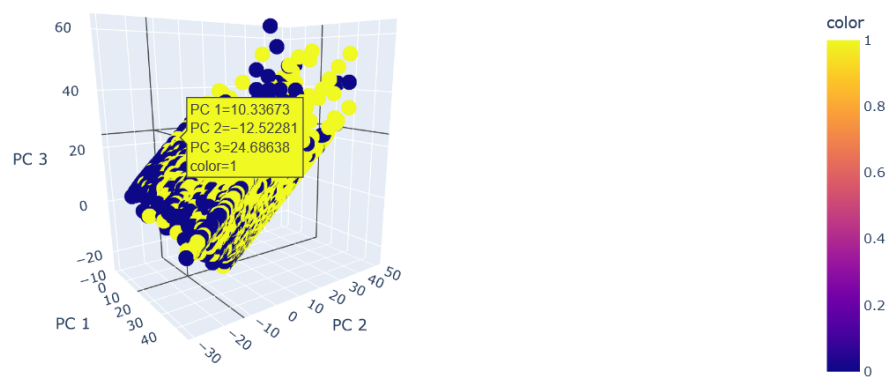
Figure 17: Information Conservée

Plus la corrélation est élevée, moins la perte d'information par projection est élevée.

3 composantes :

En projetant sur 3 composantes nous conservons 74.5% d'information, nous pouvons observer la séparation spatiale des individus ayant le diabète ou non suivant les 3 composantes.

Total Explained Variance: 74.53%



`explained_var_ratio[0.53254584 0.21277493 0.20775049]`

Figure 18: PCA 3

3 Gradient Boosting de Friedman et XGBoost

Le gradient boosting de Friedman repose sur l'idée d'ajouter des modèles de manière itérative pour corriger les erreurs du modèle précédent. Plus précisément, l'algorithme construit un modèle $F_0(x)$ à partir des données d'entraînement S , puis itère sur un nombre fixé M d'étapes pour construire des modèles $F_m(x)$, $m = 1, \dots, M$, de plus en plus complexes.

Chaque nouveau modèle $F_m(x)$ est construit pour minimiser la fonction de coût $l(y, F_{m-1}(x) + F_m(x))$, où y est la variable cible et l est une fonction de perte dépendant du type de tâche (classification ou régression). $F_{m-1}(x)$ est la prédiction du modèle combiné jusqu'à l'étape $m - 1$. Ainsi, le nouveau modèle est construit pour prédire les résidus $r_m = y - F_{m-1}(x)$, c'est-à-dire la différence entre les vraies valeurs et les prédictions du modèle combiné à l'étape $m - 1$. Plus formellement, le modèle $F_m(x)$ est construit en minimisant la fonction de perte de la forme :

$$L^{(t)} = \sum_i l(\hat{y}_i^{(t-1)} + F_t(x_i), y_i) + \Omega(F_t)$$

où N est le nombre de données d'entraînement, l est la fonction de perte, $\Omega(F_m)$ est une fonction de régularisation qui mesure la complexité du modèle $F_m(x)$, et $F_{m-1}(x)$ est la prédiction du modèle combiné jusqu'à l'étape $m - 1$. Le modèle final est obtenu en combinant tous les modèles construits à chaque étape :

$$F(x) = F_0(x) + \sum_{m=1}^M \gamma_m F_m(x)$$

où γ_m est le coefficient d'apprentissage, qui contrôle l'importance du modèle $F_m(x)$ dans la prédiction finale.

L'algorithme de gradient boosting de Friedman utilise également une technique appelée descente de gradient stochastique (SGD) pour optimiser la fonction de perte. SGD consiste à utiliser des échantillons aléatoires de données d'entraînement pour mettre à jour les paramètres du modèle à chaque étape. Cette technique est efficace pour traiter de grands ensembles de données et éviter le sur-apprentissage.

3.1 XGBoost objectif d'apprentissage généralisé

Définitions:

On définit l'ensemble d'apprentissage :

$$\mathcal{S} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

Et, soit également l'ensemble des arbres de régression \mathcal{F} . Nous avons :

$$\mathcal{F} = \{f(x) = \omega_{q(x)}\}$$

où $q : \mathbf{R}^m \rightarrow T$ est la fonction qui à une observation associe la feuille de l'arbre de taille T .

On construit le classifieur \hat{y}_i de la façon suivante :

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^K f_k(x_i), f_k \in \mathcal{F}$$

Avec K le nombre d'arbres créés et f_k les différents arbres de régression.

On construit l'estimation grâce à une somme des estimations sur l'ensemble de K arbres.

L'objectif régularisé du classifieur ϕ comme construit ci dessus est exprimé par :

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

On prend l une fonction coût convexe, et on somme sur les différentes prédictions $\Omega(f_k) = \gamma T + \frac{1}{2} \lambda \|\omega\|^2$, où γ et λ sont des paramètres.

Intuitivement on pénalise la complexité de chaque arbre f_k de par sa taille et la norme de ses poids, de cette manière on cherche à contrôler le sur-apprentissage.

3.2 Algorithme XGBoost

Le modèle précédent a comme paramètre des fonctions et ne peut être optimisé en utilisant des algorithmes classiques dans un espace euclidien.

Considérons $\hat{y}_i^{(t)}$ la prédiction de la i ème donnée au bout de t itérations. nous devons ajouter f_t pour minimiser l'objectif suivant :

$$\mathcal{L}^{(t)} = \sum_i l(\hat{y}_i^{(t-1)} + f_t(x_i), y_i) + \Omega(f_t)$$

cela signifie qu'on ajoute f_t ajustant le mieux le modèle suivant l'équation précédente à chaque itération, par un développement limité on obtient :

$$\begin{cases} \tilde{\mathcal{L}}^{(t)} &= \sum_i l(\hat{y}_i^{(t-1)} + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 + \Omega(f_t) \\ g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)}) \end{cases} \quad (3)$$

en supposant qu'on a minimisé pour chaque étape $1, \dots, t-1$ on a alors:

$$\operatorname{argmin}_{f_t} \tilde{\mathcal{L}}^{(t)} = \operatorname{argmin}_{f_t} \sum_i g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 + \gamma T + \frac{1}{2} \lambda \sum_{k=1}^T \omega_k^2$$

On pose : $I_j = \{i, q(x_i) = j\}$ on peut alors réécrire :

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^T \left(\sum_{i \in I_j} g_i \right) \omega_j + \frac{1}{2} \sum_{i \in I_j} (h_i + \lambda) \omega_j^2 + \gamma T$$

La somme porte désormais sur un seul indice. On cherche à minimiser suivant ω_j la fonction $\tilde{\mathcal{L}}^{(t)}$ on cherche à annuler le gradient. On a :

$$\frac{\partial}{\partial \omega_j} \tilde{\mathcal{L}}^{(t)} = \frac{\partial}{\partial \omega_j} \left(\sum_{j=1}^T \left(\sum_{i \in I_j} g_i \right) \omega_j + \frac{1}{2} \sum_{i \in I_j} (h_i + \lambda) \omega_j^2 + \gamma T \right)$$

Comme les termes ne dépendent pas de ω_j alors ils s'annulent et on obtient donc :

$$\frac{\partial}{\partial \omega_j} \left(\sum_{i \in I_j} g_i \right) \omega_j + \frac{1}{2} \sum_{i \in I_j} (h_i + \lambda) \omega_j^2 + \gamma T = \sum_{i \in I_j} g_i + \left(\sum_{i \in I_j} h_i + \lambda \right) \omega_j$$

Le gradient s'annule donc pour : $\omega_j = - \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$

En remplaçant cette valeur dans la fonction objectif on obtient :

$$\boxed{\tilde{\mathcal{L}}^{(t)} = -\frac{1}{2} \sum_{j=1}^T \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T}$$

3.3 Différences XGB /Gradient Boosting de Friedman

1. Fonction de coût : Le gradient boosting de Friedman utilise une fonction de coût différentiable, généralement une fonction de perte de régression ou de classification, pour minimiser les erreurs du modèle. En revanche, XGBoost utilise une fonction de coût plus générale appelée fonction d'objectif régularisée, qui peut inclure des termes de régularisation pour contrôler la complexité du modèle et prévenir le sur-apprentissage.
2. Fonction de régularisation : Le gradient boosting de Friedman utilise une fonction de régularisation appelée "shrinkage", qui consiste à réduire les coefficients d'apprentissage des modèles successifs pour éviter le sur-apprentissage. XGBoost utilise également la fonction de régularisation de shrinkage, ainsi qu'une autre fonction de régularisation appelée "L1/L2 regularization" pour pénaliser les coefficients des modèles et favoriser la sélection de fonctionnalités importantes.
3. Efficacité : XGBoost est généralement plus rapide que le gradient boosting de Friedman car il utilise une mise en œuvre parallèle pour construire les modèles successifs. XGBoost est également conçu pour gérer des ensembles de données de grande taille et haute dimensionnalité, tandis que le gradient boosting de Friedman peut être plus susceptible de sur-apprentissage sur de tels ensembles de données.
4. Optimisation : XGBoost utilise une optimisation basée sur un arbre pour construire les modèles successifs, ce qui permet de gérer efficacement des données de grande dimensionnalité et de haute complexité. Le gradient boosting de Friedman, en revanche, utilise une optimisation basée sur une descente de gradient stochastique, ce qui peut prendre plus de temps pour converger sur des ensembles de données complexes.

Le modèle XGBoost est une implémentation du gradient boosting de Friedman optimisé et avec une régularisation spécifique limitant la complexité des arbres.

3.4 Application : Modèle de Prédiction de Diabète

A partir d'un jeu de données nous voulons prévenir le risque de diabète, les variables observées sont : l'imc, l'âge le sexe, fumeur ou non etc... Nous voulons procéder à une tâche de classification pour appliquer l'algorithme XGBoost.

3.4.1 Définition de l'objectif et métriques associées

Afin d'optimiser notre travail il est important de définir nos objectifs vis-à-vis de la classification, étant confrontés à une problématique de santé il peut être important de minimiser les faux négatifs: les personnes prédites comme non à risque pour lequel le modèle se trompe. Pour cela nous disposons des métriques suivantes :

$$Accuracy = \frac{\text{Bonnes predictions}}{\text{Nombre total de prediction}}$$

On décompose les résultats de la manière suivante :

TP : True positive, vrais positifs prédiction 1 juste.

TN : True negative, vrais négatifs prédiction 0 juste.

FP : False positive, faux positif prédiction 1 fausse.

FN : False negative, faux négatifs prédiction 0 fausse.

On a alors les définitions suivantes :

$$Recall = \frac{TP}{TP + FN}$$
$$Precision = \frac{TP}{TP + FP}$$

3.4.2 Sélection de variables via Feature Importance

Afin d'observer l'importance des variables il est intéressant de faire tourner un arbre de classification et de visualiser les règles de décisions et variables associées.

Nous utilisons l'algorithme de scikit-learn qui utilise par défaut la règle de décision gini comme décrite précédemment.

Ici nous pouvons voir qu'avec un arbre de profondeur 2 les variables utilisées sont: "HighBP" et "GenHlth".

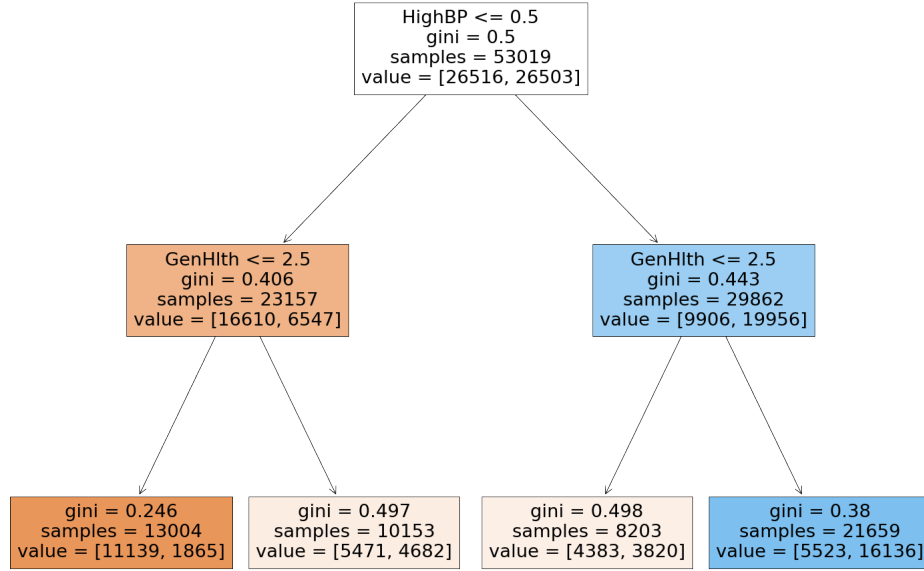


Figure 19: Arbre Construit vit l'indice de Gini

Soit un ensemble de données d'entraînement $\mathcal{D} = (x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$, où $x_i = (x_{i_1}, x_{i_2}, \dots, x_{i_n})$ est un vecteur de n caractéristiques ou variables pour chaque exemple i et y_i est la variable cible associée.

L'algorithme du XGBoost va venir construire un ensemble de modèles d'arbres de décision en utilisant le boosting vu précédemment. Chaque arbre de décision est entraîné sur un sous-ensemble des données d'entraînement et une sous-ensemble des caractéristiques. Les arbres de décision sont ajoutés à l'ensemble en séquence, chacun cherchant à corriger les erreurs résiduelles du modèle précédent.

A chaque caractéristique j est calculé l'importance relative par une somme pondérée de la réduction d'erreur à chaque fois qu'on utilise cette caractéristique pour diviser les données dans tous les arbres de décision de tous les modèles. On utilisera comme vu précédemment la mesure de Gini.

Alors, pour chaque caractéristique j , nous calculons le score d'importance I_j comme ceci :

$$I_j = \sum_{t=1}^T w_{j,t} r_t \quad (4)$$

où $w_{j,t}$ est le poids de la caractéristique j dans le t -ème arbre de décision et r_t est la réduction de l'erreur obtenue en utilisant cette division.

On trouve $w_{j,t}$ en comptant le nombre de fois où la caractéristique j est utilisée pour diviser les données dans le t -ème arbre de décision par rapport à la proportion des exemples d'entraînement atteignant chaque branche de la division.

On peut donc normaliser I_j en divisant par la somme des scores d'importance de toutes les caractéristiques, afin de donner une mesure relative de l'importance de chaque caractéristique.

Soit I_j le score d'importance de la caractéristique j calculé comme décrit précédemment, et I l'ensemble des scores d'importance pour toutes les caractéristiques $j \in 1, 2, \dots, n$.

On a alors :

$$I'_j = \frac{I_j}{\sum_{j=1}^n I_j}$$

Le score d'importance normalisé I'_j représente ainsi la proportion de la contribution de la caractéristique j à l'ensemble de modèles.

Ici, nous allons venir calculer le feature importance de chaque variable par rapport au Diabète pour le classifieur XGBoost.

Il faut bien remarquer qu'en fonction du classifieur utilisé, les features importances évolueront également.

Pour le Classifieur XGBoost, nous obtenons alors :

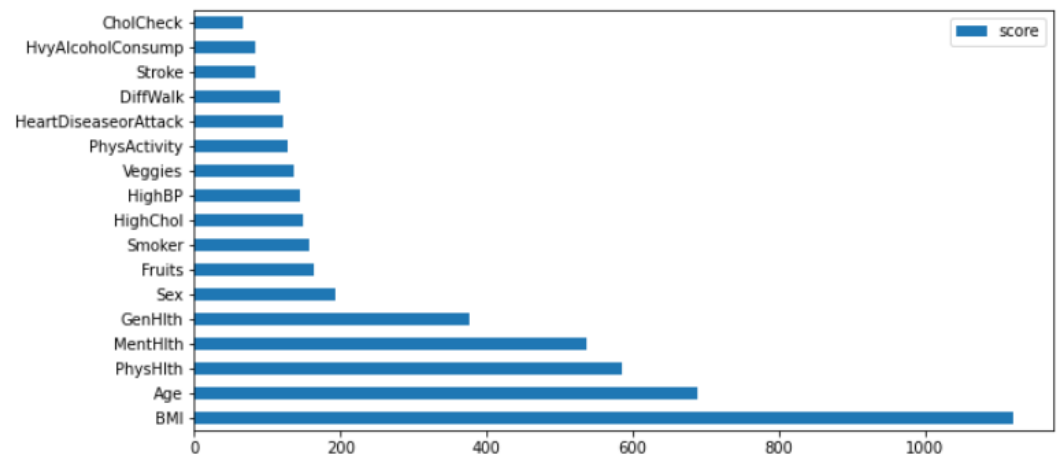


Figure 20: Feature Importance via XGBoost

L'importance d'une variable est calculée comme étant la réduction totale du critère (ici gini) normalisée amenée par cette variable.

3.4.3 Modèles xgboost

Nous allons dans cette partie optimiser les hyperparamètres de l'algorithme xgboost à l'aide de courbes de validation et de recherche par grille de paramètres.

1. Courbe de validation :

Afin d'optimiser nos hyperparamètres, nous pouvons étudier l'évolution des scores d'entraînement et de test en faisant évoluer un par un les paramètres.

Gamma :

On rappelle :

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

On prend l une fonction coût convexe, et on somme sur les différentes prédictions $\Omega(f_k) = \gamma T + \frac{1}{2} \lambda \|\omega\|^2$ γ, λ étant des paramètres.

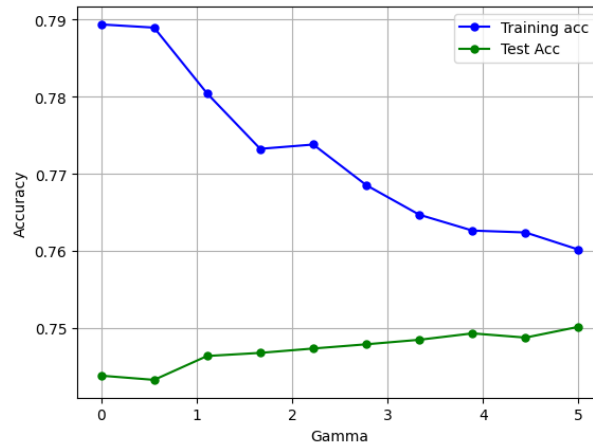


Figure 21: Validation Curve pour le paramètre Gamma

En augmentant le paramètre gamma, on constate une diminution du score d'apprentissage et une augmentation du score de test, ce qui est cohérent avec la nature de l'hyper-paramètre gamma, il permet de réguler le surapprentissage en limitant la complexité du modèle.

Reg lambda :

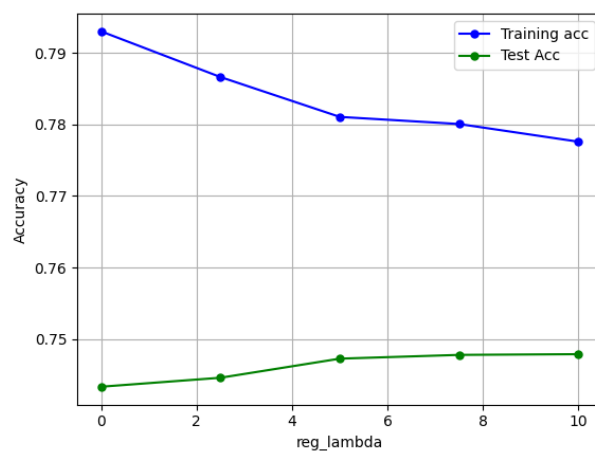


Figure 22: Validation Curve pour le paramètre Lambda

Un autre paramètre de régularisation est lambda : La sensibilité de ce paramètre est moins importante que gamma, cependant, on peut faire le constat attendu de limitation du sur apprentissage.

Learning rate :

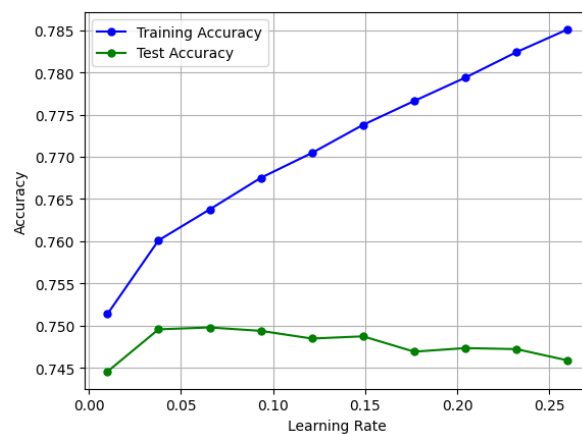


Figure 23: Validation Curve pour le paramètre Learning Rate

Un autre paramètre qui n'apparaît pas directement dans la formule de coût est le learning-rate, nous constatons une convergence de la courbe d'apprentissage dépendante linéairement par rapport au learning rate.

2. Recherche par grille

Soit des hyperparamètres θ , les hyperparamètres que nous nous devons de prédéfinir à l'avance.

Le grid search consiste à spécifier une grille Θ de valeurs possibles pour chaque hyperparamètre et à évaluer la performance du modèle pour chaque combinaison de valeurs. Autrement dit, on vient effectuer toutes les combinaisons possibles de nos hyperparamètres et on peut évaluer la performance de chaque combinaison à l'aide d'une validation croisée.

La méthode de validation croisée consiste à diviser l'ensemble de données en k sous-ensembles (ou plis), en utilisant $k-1$ sous-ensembles pour l'entraînement et le reste pour la validation. Cette procédure est répétée k fois, chaque sous-ensemble étant utilisé une fois pour la validation. Le modèle est alors évalué en prenant la moyenne des performances acquises pour chaque pli.

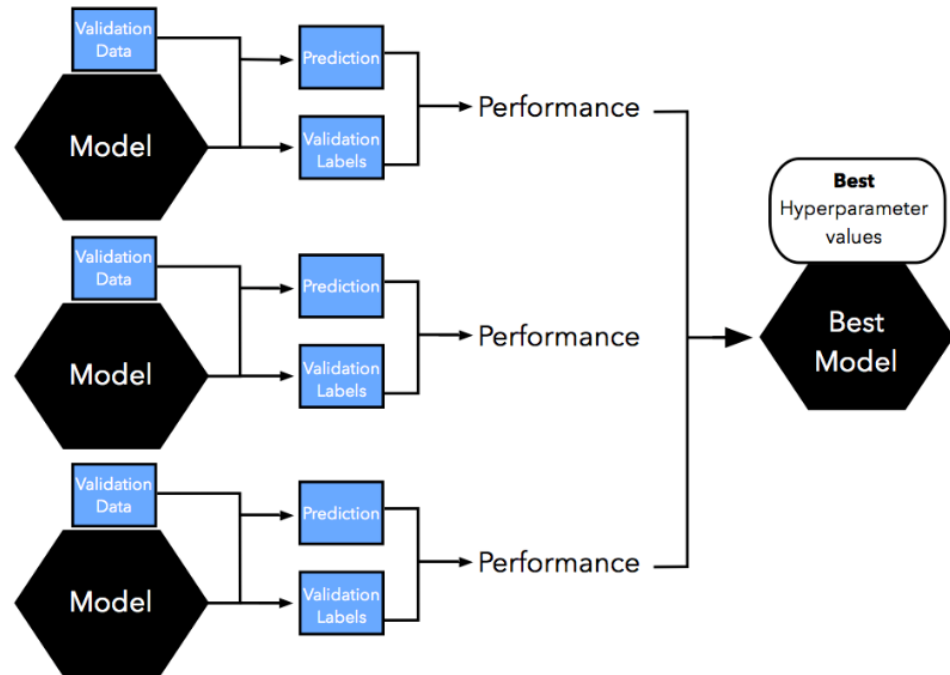


Figure 24: Visualisation du Fonctionnement de la Recherche par Grille

Alors, la fonction de performance $f(\theta)$ est la moyenne de la performance du modèle sur les k plis.

On l'écrit comme ceci :

$$f(\theta) = \frac{1}{k} \sum_{i=1}^k \text{performance}(\theta, \text{pli}_i) \quad (5)$$

où $\text{performance}(\theta, \text{pli}_i)$ est la performance du modèle pour la combinaison d'hyperparamètres θ sur le pli i .

Lorsque nous avons fini de tester toutes les combinaisons, le grid search retourne le modèle contenant les meilleurs hyperparamètres au vue de la métrique choisie.

On recherche alors θ^* des hyperparamètres qui maximise la fonction de performance $f(\theta)$, avec θ est notre vecteur d'hyperparamètres.

On a alors :

$$\theta^* = \arg \max_{\theta \in \Theta} f(\theta) \quad (6)$$

En revanche, nous avons fait face à un problème pour avoir la meilleur optimisation possible de nos modèles. En effet, le grid search est très couteux en temps d'exécution et RAM utilisée, et, avec la puissance de nos ordinateurs seulement, nous ne pouvons pas gérer des milliers de fit croisés. De plus, le grid search est toujours très intéressant mais ne propose pas de point de vue visuel (graphique par exemple).

Néanmoins, nous avons quand même pu l'opérer pour un nombre de fit restreint, mais qui améliorera quand même notre modèle.

Nous pouvons fixer un ensemble de paramètres à parcourir à l'aide d'un dictionnaire en python :

```
param_grid = {
    'max_depth': [2, 3],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 200],
    'gamma': [1, 3, 5],
    'subsample': [0.8, 0.9],
    'colsample_bytree': [0.8, 0.9, 1]
}
```

Pour cette grille nous obtenons les résultats suivants

”Meilleurs hp : {'colsample_bytree': 0.8, 'gamma': 3, 'learning_rate': 0.1,
Performance du mod : 0.7562240663900415”

3.4.4 Evaluation de modèle

Après entraînement de notre modèle nous pouvons évaluer suivant plusieurs critères notre modèle :

- accuracy nombre de faux négatifs...
- dépendance du modèle à la quantité de donnée

Matrice de confusion :

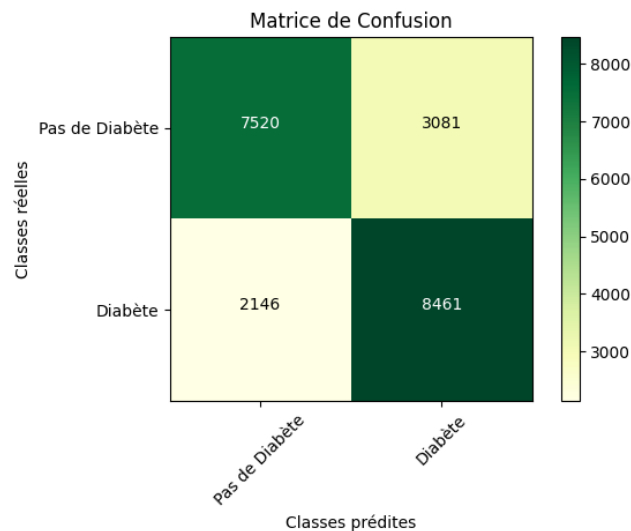


Figure 25: Matrice de Confusion pour le modèle XGBoost optimisé

Nous pouvons constater ici qu'on a :

$$TP = 8461$$

$$TN = 7520$$

$$FP = 3081$$

$$FN = 2146$$

On a alors :

$$Recall = \frac{8461}{8461 + 2146} = 0.79...$$

$$Precision = \frac{8461}{8461 + 3081} = 0.73...$$

La proportion de faux négatifs dans notre modèle est faible, ce qui remplit notre objectif initial.

Courbe d'apprentissage :

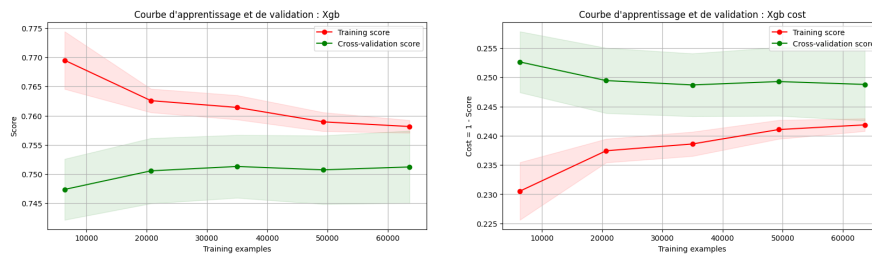


Figure 26: Courbe d'apprentissage du XGBoost

L'objectif de la courbe d'apprentissage est d'obtenir les scores des modèles entraînés sur différents volumes de données, ici, nous pouvons constater que le score par cross-validation augmente entre 10000 et 20000 données mais pas de différence significative au delà, le score d'apprentissage diminue avec la quantité de données.

Pour ce modèle et ces paramètres il ne semble pas nécessaire d'obtenir plus de données, nous pourrions donc conclure que les associations statistiques entre les variables explicatives et la variable diabète sont insuffisantes pour converger plus.

Notons que pour une quantité de données différente, il serait nécessaire de rechercher d'autres paramètres.

Nous allons comparer les résultats :

1. Du modèle XGBoost entraîné sur la totalité des données.
2. Du modèle XGBoost avec selection de features : PCA

	Model	Time	Accuracy	Precision	Recall	F1-Score	AUC
0	XGBoost	6.471033	0.751367	0.732560	0.792024	0.761133	0.751356
1	XGBoost_pca3	3.787049	0.694313	0.677851	0.740926	0.707986	0.694300
2	XGBoost_pca5	7.872873	0.733733	0.714088	0.779862	0.745527	0.733719

Figure 27: Scores comparés

Les résultats sont cohérents. En effet, lorsque nous prenons seulement 3 composantes dans le PCA, il en résulte un temps de calcul presque divisé par 2 mais nous obtenons une AUC et une Accuracy générale bien inférieure. Nous savons d'après l'étude précédente sur le PCA qu'avec seulement 3 composantes, notre jeu de données transformé à l'aide du PCA perd en information (ne couvre que 74,5% de la variance totale).

Le meilleur modèle reste le XGBoost, avec des scores de plus haute précision que les deux autres modèles.

4 Etude du LightGBM

Nous nous rendons compte après l'utilisation du XGBoost que certaines limitations existent et qu'il serait difficile d'y palier en optimisant seulement l'algorithme du XGBoost.

En effet, XGBoost est extrêmement long à entraîner des modèles sur de grandes quantités de données. Comme l'algorithme traite l'intégralité de l'ensemble d'apprentissage à chaque itération, avec un jeu de données pouvant contenir des millions de variables ou catégories, nous comprenons qu'il va falloir compenser ces problèmes.

Pour cela, l'algorithme du LightGBM propose deux solutions nouvelles contrecarrant ces soucis :

L'algorithme du Gradient-based One-Side Sampling et l'algorithme Exclusive Feature Bundling.

Nous allons maintenant voir plus en détail ce que sont ces algorithmes et comment ils palient au manque de vitesse de calcul du XGBoost.

4.1 Algorithme GOSS

L'algorithme GOSS est une technique d'échantillonnage intelligent permettant à LightGBM de sélectionner les exemples d'apprentissage les plus utiles pour chaque itération, plutôt que de devoir regarder l'intégralité de l'ensemble d'apprentissage.

Alors, LightGBM entraînera des modèles plus rapidement que XGBoost, tout en maintenant une précision élevée. Pour expliquer plus en détail, avec l'algorithme GOSS, nous allons :

1. **Conserver** toutes les instances avec de grands gradients et effectue un échantillonnage aléatoire sur les instances avec de petits gradients.
2. **Tri** des instances de données selon la valeur absolue de leurs gradients et sélection les meilleures $a * 100\%$
3. **Echantillonner** de manière aléatoire $b * 100\%$ instances parmi le reste des données.
4. **Amplifier** les données échantillonnées avec de petits gradients par une constante $1 - a/b$ lors du calcul du gain d'information.

L'algorithme GOSS peut être décrit mathématiquement comme suit :

Entrées : $\mathcal{S} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$: ensemble de données d'entraînement,
 d : nombre d'itérations, a : taux d'échantillonnage des données à grand gradient, b : taux d'échantillonnage des données à petit gradient, ℓ : fonction de perte, L : modèle d'apprentissage faible.

$modèles \leftarrow , fact \leftarrow 1 - \frac{a}{b}$
 $topN \leftarrow a \times n, randN \leftarrow b \times n$

Pour $i = 1$ à d **faire** :

1. $\hat{y} \leftarrow$ prédiction de L sur \mathcal{I}
2. $g \leftarrow \nabla_{\hat{y}} \ell(y, \hat{y})$
3. $w \leftarrow 1, 1, \dots, 1$
4. $triée \leftarrow$ indices triés par ordre décroissant de $|g|$
5. $topSet \leftarrow triée[1 : topN]$
6. $randSet \leftarrow$ échantillonnage aléatoire de $triée[topN : n]$ avec taille $randN$
7. $usedSet \leftarrow$ concaténation de $topSet$ et $randSet$
8. $w[randSet] \leftarrow w[randSet] \times fact$ pour affecter le poids $fact$ aux données à petit gradient
9. $nouveauModèle \leftarrow L(\mathcal{I}[usedSet], -g[usedSet], w[usedSet])$
10. $modèles \leftarrow modèles \cup nouveauModèle$

4.2 Algorithme du Exclusive Feature Bundling

L'Exclusive Feature Bundling (EFB) est une technique de régularisation utilisée dans l'algorithme LightGBM pour regrouper des caractéristiques similaires afin de réduire la dimensionnalité des données.

L'EFB fonctionne en utilisant une matrice de similarité qui mesure la similarité entre les caractéristiques. Les caractéristiques similaires ont une valeur de similarité élevée, tandis que les caractéristiques qui sont différentes ont une valeur de similarité faible. Les caractéristiques avec des valeurs de similarité élevées sont ensuite regroupées dans des ensembles exclusifs (bundles). Nous avons plusieurs versions de ce procédé :

- Greedy Bundling
- Merge Exclusive Features

Greedy Bundling : L'algorithme Greedy Bundling est un algorithme de regroupement de caractéristiques pour la réduction de la dimensionnalité. L'objectif est de regrouper les caractéristiques similaires dans un ensemble exclusif appelé bundle. Les bundles exclusifs peuvent être utilisés pour construire des modèles plus simples, plus interprétables et plus efficaces.

L'algorithme Greedy Bundling utilise une matrice de similarité pour mesurer la similarité entre les caractéristiques. La matrice de similarité est définie comme une matrice symétrique où chaque élément est la mesure de similarité entre deux caractéristiques.

Entrée :

- Matrice de données \mathbf{X} de taille $n \times p$
- Matrice de similarité \mathbf{S} de taille $p \times p$ (exemple corrélation de Pearson).
- Seuil de similarité t

Sortie : Liste des bundles exclusifs \mathbf{B}

- Initialiser chaque caractéristique i comme un bundle exclusif $b_i = i$
- Pour chaque paire de caractéristiques i, j avec $i < j$
- Si $\mathbf{S}_{ij} \geq t$ alors fusionner les bundles exclusifs b_i et b_j : $b_i \leftarrow b_i \cup b_j, b_j \leftarrow b_i$
- Retourner la liste des bundles exclusifs \mathbf{B}

Algorithm 1 Exclusive Feature Bundling

Matrice de données \mathbf{X} de taille $n \times p$, matrice de similarité \mathbf{S} de taille $p \times p$,
seuil de similarité t Liste des bundles exclusifs \mathbf{B}

```
for  $i = 1$   $p$  do Initialiser chaque caractéristique  $i$  comme un bundle exclusif  
 $b_i = i$ ;  
    for  $i = 1$   $p - 1$  do  
        for  $j = i + 1$   $p$  do  
            if  $S_{ij} \geq t$  then Fusionner les bundles exclusifs  $b_i$  et  $b_j$ :  $b_i \leftarrow b_i \cup b_j$ ,  
 $b_j \leftarrow b_i$ ; return Liste des bundles exclusifs  $\mathbf{B}$ 
```

En résumé, l’algorithme Greedy Bundling est un algorithme de regroupement de caractéristiques qui utilise une matrice de similarité pour trouver des paires de caractéristiques similaires et les fusionner en bundles exclusifs. Cette technique de réduction de la dimensionnalité peut améliorer les performances des modèles en simplifiant les caractéristiques et en améliorant la capacité de généralisation.

Merge Exclusive Features

Entrée :

- Ensemble de données d’apprentissage D
- Liste des bundles exclusifs \mathbf{B}

Sortie : Un arbre de décision \mathcal{T}

1. Pour chaque ensemble exclusif $b \in \mathbf{B}$, calculer la variance intra-bundle σ_b^2 ;
2. Calculer la variance globale σ_{global}^2 de l’ensemble de données D ;
3. Si la variance globale σ_{global}^2 est inférieure à un seuil prédéfini, arrêter la récursion et retourner une feuille avec la valeur moyenne des étiquettes des données D ;
4. Trouver le meilleur bundle exclusif b pour diviser les données D ;
5. Pour chaque sous-ensemble de données D' qui est obtenu après la division de D par b , répéter les étapes 1-4 récursivement pour construire un sous-arbre de \mathcal{T} ;

Algorithm 2 Merge Exclusive Features

Matrice de données \mathbf{X} de taille $n \times p$, matrice de similarité \mathbf{S} de taille $p \times p$, seuil de similarité t , paramètre de régularisation λ , nombre maximum d'itérations T Ensemble optimal de bundles exclusifs \mathbf{B} $\mathbf{B}^{(0)} \leftarrow \text{ExclusiveFeatureBundling}(\mathbf{X}, \mathbf{S}, t); \mathbf{B} \leftarrow \mathbf{B}^{(0)}$;

for $t = 1$ T **do**

for $k = 1$ $|\mathbf{B}|$ **do** Calculer $\mathbf{w}^{(k)}$ par la régression linéaire de \mathbf{y} sur les caractéristiques dans le bundle \mathbf{B}^k en utilisant la régularisation L2 avec un paramètre λ ;

for $i = 1$ p **do** Calculer la contribution de chaque bundle \mathbf{B}^k à la réduction d'erreur pour la caractéristique i :

$$\Delta_{\text{err}}(\mathbf{B}^k, i) = \frac{1}{n} \sum_{j=1}^n \left(y_j - \sum_{k' : i \in \mathbf{B}^{k'}} \mathbf{w}^{(k')} \mathbf{x}_{j,i} \right)^2 - \frac{1}{n} \sum_{j=1}^n \left(y_j - \sum_{k' : i \in \mathbf{B}^{(0)}} \mathbf{w}^{(k')} \mathbf{x}_{j,i} \right)^2 \quad (7)$$

où $\mathbf{x}_{j,i}$ est la valeur de la caractéristique i de l'observation j ; Pour chaque bundle \mathbf{B}^k , trouver la caractéristique i_k qui maximise $\Delta_{\text{err}}(\mathbf{B}^k, i)$;

if le gain de performance est nul **then** Arrêter l'algorithme et retourner \mathbf{B} ; Mettre à jour \mathbf{B} en remplaçant la caractéristique i_k par un nouveau bundle exclusif $\mathbf{B}_k^{(t)}$ obtenu en appliquant l'algorithme ExclusiveFeatureBundling sur les caractéristiques restantes; **return** \mathbf{B}^*

4.3 Algorithme LightGBM

Dans le cadre que l'on s'est donné pour l'étude du Diabète, on sera donc confronté à de la classification binaire.

L'algorithme qui suit est donc la version classification binaire du LightGBM. La seule différence est que nous prendrons ici la fonction de perte de log vraisemblance binaire pour itérer dans l'algorithme.

Entrée:

- Soit $D = (\mathbf{x}_i, \mathbf{y}_i)_{i=1}^n$ l'ensemble de nos données d'apprentissage, où \mathbf{x}_i est un vecteur de caractéristiques et y_i est l'étiquette de classe binaire (0 ou 1) de l'exemple i .

- On prend M le nombre d'arbres de décision.

- On prend L le nombre de feuilles maximales par arbre de décision.

- On choisit la profondeur maximale de l'arbre de décision D .

- On choisit η le taux d'apprentissage.

- On choisit la fonction de perte $L(y, f)$, ici fonction de perte de log vraisemblance binomiale.

- Fonction de mesure de performance, telle que l'exactitude ou l'AUC.

Sortie :

- Ensemble de modèles de gradient boosting $f_{m=1}^M$.

1. Initialiser les prédictions $f_0(x) = \log\left(\frac{\sum_{i=1}^n y_i}{\sum_{i=1}^n (1-y_i)}\right)$, où \log est la fonction logarithme naturel.

2. **Pour** chaque arbre $m = 1, \dots, M$:

(a) Calculer le gradient $g_{i_m} = \frac{\partial L(y_i, f_{m-1}(\mathbf{x}_i))}{\partial f_{m-1}(\mathbf{x}_i)}$ et le hessien $h_{i_m} = \frac{\partial^2 L(y_i, f_{m-1}(\mathbf{x}_i))}{\partial f_{m-1}(\mathbf{x}_i)^2}$ pour chaque exemple i .

(b) Entraîner un arbre de décision T_m en utilisant l'ensemble de données $(\mathbf{x}_i, g_{i_m}, h_{i_m})_{i=1}^n$ avec les paramètres L , D , η et les caractéristiques les plus importantes sélectionnées lors des étapes précédentes.

(c) **Pour** chaque feuille j de l'arbre T_m , calculer la prédiction $\gamma_{jm} = -\frac{\sum_{i \in \text{Leaf}(j)} g_{im}}{\sum_{i \in \text{Leaf}(j)} h_{im} + \lambda}$, où $\text{Leaf}(j)$ est l'ensemble des exemples dans la feuille j et λ est un paramètre de régularisation.

(d) Mettre à jour les prédictions de l'ensemble de modèles : $f_m(x) = f_{m-1}(x) + \eta \sum_{j=1}^J \gamma_{jm} 1_{x \in \text{Leaf}(j)}$, où J est le nombre de feuilles dans l'arbre T_m et $1_{x \in \text{Leaf}(j)}$ est une fonction indicatrice qui vaut 1 si l'exemple x appartient à la feuille j et 0 sinon.

3. **Renvoyer** l'ensemble de modèles $f_{m=1}^M$.

4.4 Modèle LightGBM

Comme dans la partie précédente nous venons ici faire de la recherche d'hyperparamètres cette fois ci orienté par le LightGBM.

1. Recherche par Grid Search :

Le principe reste exactement le même que dans le cas du XGBoost, autant nous nous référerons et utiliserons sans redémontrer le Grid Search en analyse uniquement.

Avec LightGBM, les hyperparamètres à optimiser seront différent que dans le XGBoost.

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
Meilleurs hyperparamètres : {'colsample_bytree': 0.5, 'learning_rate': 0.05, 'n_estimators': 200, 'reg_alpha': 0.01, 'reg_lambda': 0, 'subsample': 0.7}
Accuracy LGBM : 0.7569784986797435
```

Figure 28: Grid Search

Comme nous a conseillé Mr.Cohen, mieux vaut faire plusieurs Grid Search contenant moins de paramètres à tester mais en réduisant l'intervalle de recherche à chaque fois. Nous voyons qu'en reprécisant nos hyperparamètres par rapport à ceux obtenus ci dessus :

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Meilleurs hyperparamètres : {'colsample_bytree': 0.5, 'learning_rate': 0.05, 'n_estimators': 200, 'num_leaves': 40, 'reg_alpha': 0.005, 'reg_lambda': 0}
Accuracy LGBM : 0.7570256506978499
```

Figure 29: Grid Search Final

4.5 Evaluation de notre modèle LightGBM

Différents critères vont nous servir à évaluer notre modèle comme précédemment, tel que les différents scores (Accuracy, Recall, Precision...) et nous pourrons visualiser cela grâce à la matrice de confusion de notre modèle.

Bien entendu, le modèle utilisé est le modèle tiré de la recherche par Grid Search des meilleurs hyperparamètres. Ce seront ces hyperparamètres qui seront utilisés pour le réglage du modèle.

Nous avons :

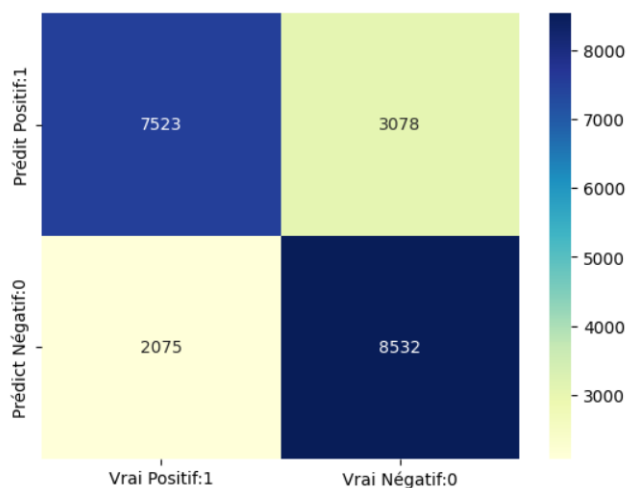


Figure 30: Matrice de Confusion pour LightGBM

On peut constater ici qu'on a :

$$TP = 8532$$

$$TN = 7523$$

$$FP = 3078$$

$$FN = 2075$$

Nous obtenons alors :

Resultats :

	precision	recall	f1-score	support
0	0.78	0.71	0.74	10601
1	0.73	0.80	0.77	10607
accuracy			0.76	21208
macro avg	0.76	0.76	0.76	21208
weighted avg	0.76	0.76	0.76	21208

Figure 31: Scores du LightGBM

```
Affichage des scores: lgbm
Scores training f1 weighted : 0.762 (+/- 0.001)
Scores test f1 weighted : 0.751 (+/- 0.005)
Scores training accuracy : 0.763 (+/- 0.001)
Scores test accuracy : 0.752 (+/- 0.005)
Scores training roc_auc : 0.844 (+/- 0.001)
Scores test roc_auc : 0.83 (+/- 0.006)
Scores training precision : 0.765 (+/- 0.001)
Scores test precision : 0.754 (+/- 0.005)
Scores training recall : 0.763 (+/- 0.001)
Scores test recall : 0.752 (+/- 0.005)
```

On s'intéresse à la sensibilité et la spécificité de notre modèle :

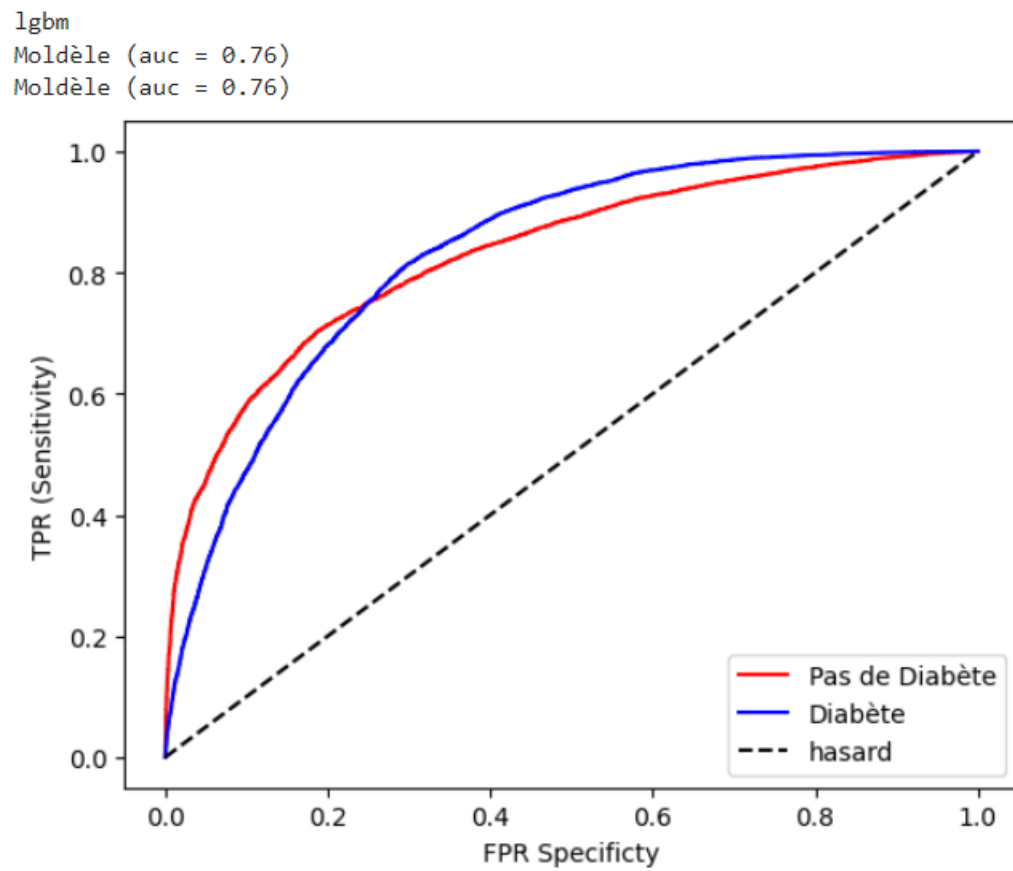


Figure 32: Spécificité du LightGBM

Regardons comment fonctionne notre modèle en fonction du nombre d'exemples d'entraînements :

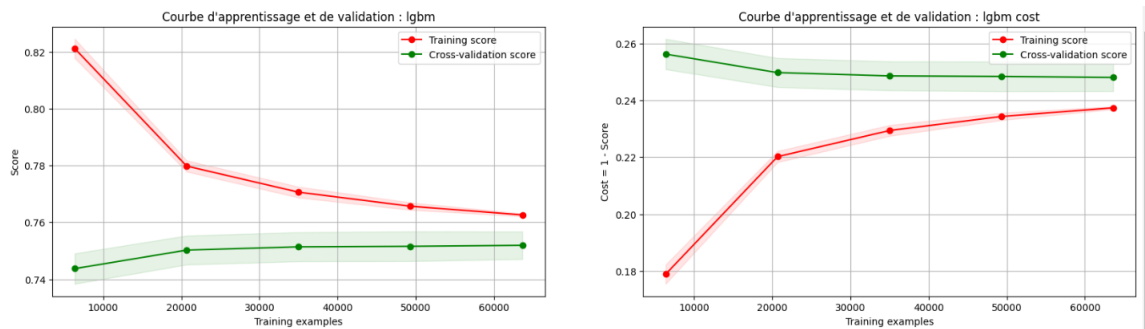


Figure 33: Courbe d'apprentissage du LightGBM

On peut aussi visualiser l'évolution des scores, notamment ici l'AUC :

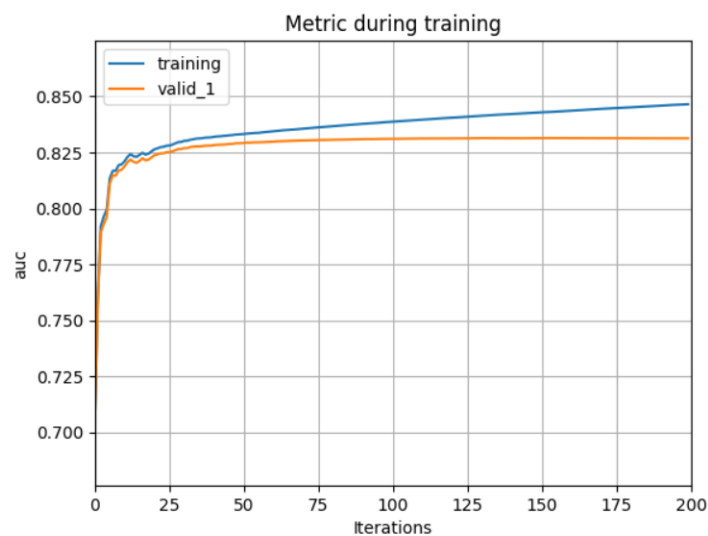


Figure 34: Score AUC par itération du LightGBM

En ce qui concerne les performances temporelles, nous nous rendons compte de la différence énorme entre le XGBoost et le LightGBM (cf Figure 35).

3	Light GBM	0.519955	0.754008	0.733253	0.798718	0.764586	0.753995
---	-----------	----------	----------	----------	----------	----------	----------

Figure 35: Temps d'exécution du LightGBM

Nous avons un temps de compilation pour le XGBoost pour le même jeu de données de 6,74 secondes, tandis que l'on descend à seulement 0,51 secondes

avec LightGBM soit plus de 13 fois plus rapide.

5 Comparaison avec d'autres modèles

5.1 Performances sur des données non structurées

En reprennant notre jeux de données de classification du risque de diabète, nous pouvons comparer avec d'autres algorithmes la performance du modèle xgboost et light gbm, en s'intéressant aussi au temps de compilation.

	Model	Time	Accuracy	Precision	Recall	F1-Score	AUC
0	Logistic Regression	1.721018	0.741796	0.727984	0.772320	0.749497	0.741787
1	Decision Trees	0.300592	0.659091	0.665946	0.638823	0.652103	0.659097
2	XGBoost	16.323769	0.751367	0.732560	0.792024	0.761133	0.751356
3	Light GBM	0.519955	0.754008	0.733253	0.798718	0.764586	0.753995

Figure 36: Scores comparatifs sur le diabète

En utilisant différent modèles de classification nous pouvons constater que l'algorithme xgboost et light GBM ont les meilleures performances suivant toutes les métriques: accuracy, precision, Recall, F1-Score et AUC, xgboost est le plus lent à entraîner, LightGBM est un des algorithmes les plus rapides et le plus performant sur cet exemple.

5.2 Application aux données non structurées

Nous avons jusque là, appliqué notre modèle xgboost et LightGBM pour de la classification avec des variables explicatives dites "non structurées", représentant des observations, nous pouvons cependant appliquer notre modèle à de la classification d'images.

Dataset Digits de Scikit Learn

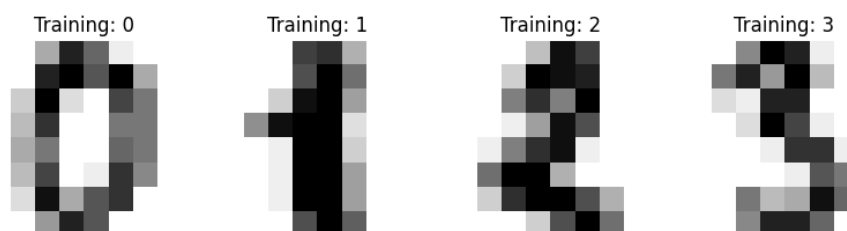


Figure 37: Dataset Digits

Réseau convolutif vs XGB

Considérons un réseau neuronal convolutif dont l'architecture est donnée ci dessous :

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 6, 6, 32)	320
conv2d_9 (Conv2D)	(None, 4, 4, 64)	18496
max_pooling2d_4 (MaxPooling 2D)	(None, 2, 2, 64)	0
dropout_6 (Dropout)	(None, 2, 2, 64)	0
flatten_3 (Flatten)	(None, 256)	0
dense_6 (Dense)	(None, 128)	32896
dropout_7 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 10)	1290
Total params: 53,002		
Trainable params: 53,002		
Non-trainable params: 0		

Figure 38: Architecture d'un réseau CNN

Accuracy CNN

Au bout de 20 epochs en 10 secondes d'itération, le score de test est de 97% avec une matrice de confusion :

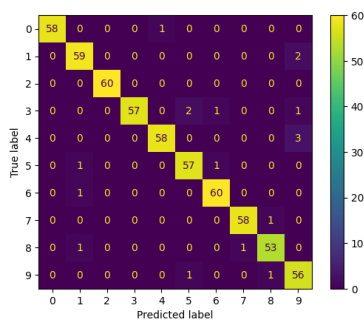


Figure 39: Matrice de confusion du réseau CNN

Accuracy XGB

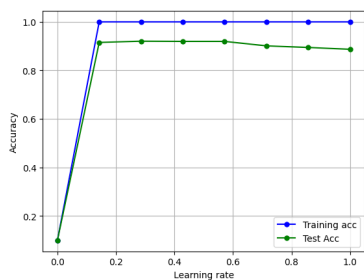


Figure 40: Validation Curve du Learning Rate

En essayant d'optimiser le learning rate et après 60 sec d'itération, l'accuracy ne dépasse pas 92%.

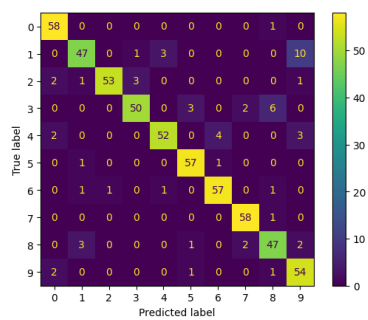


Figure 41: Matrice de confusion pour XGBoost

5.3 Conclusion :

XGBoost et LightGBM sont deux algorithmes basés sur le gradient boosting de Friedman, méthode faisant ses preuves par exemple dans des compétitions de classification ou régression sur Kaggle.

Ces deux algorithmes sont des plus efficaces sur des données tabulaires non structurées, il peuvent après recherche d'hyperparamètres, obtenir les meilleurs scores suivant toutes métriques.

LightGBM est une implémentation plus optimisée du gradient boosting, notamment grâce aux méthodes GOSS et l'exclusive feature Bundling, il sera adapté à des données de grandes dimensions de type sparse.

Cependant, ils ne sont pas toujours les plus adaptés aux données structurées, XGBoost a un temps d'entraînement qui peut être moins avantageux que d'autres algorithmes.

Les deux algorithmes sont des modèles difficile d'interprétation des règles de décision, contrairement aux arbres simples, de la régression linéaire ou encore d'autres modèles.

Pour conclure, l'étude des différents Algorithmes et la mise en place de ces derniers s'est révélée très intéressante, et les résultats sont cohérents en vue des différents outils et résultats théoriques attendus.

Nous vous remercions pour le temps que vous avez accordé à lire notre projet.

6 Sources et références

1. Gradient Boosting Jerome H. Friedman <https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-5/Greedy-function-approximation-A-gradient-boosting-machine/10.1214/aos/1013203451.full>
2. Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System.” In: KDD. Ed. by Balaji Krishnapuram et al. ACM, 2016, pp. 785–794. <http://dblp.uni-trier.de/db/conf/kdd/kdd2016.html#ChenG16>.
3. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.feature_importances_
4. PCA : <https://dridk.me/analyse-en-composante-principale.html>
5. Ada Boost Scikit Learn : <https://scikit-learn.org/stable/modules/ensemble.html#adaboost>
6. LightGBM: A Highly Efficient Gradient Boosting Decision Tree <https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html>
7. Yoav Freund and Robert E. Schapire - A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting :<https://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic-generalization.pdf>
8. <https://www.tibco.com/fr/reference-center/what-is-analysis-of-variance-anova>
9. Grid Search - Scikit Learn https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
10. Sebastian Raschka <https://sebastianraschka.com/blog/2016/model-evaluation-selection-part3.html>
11. Bibmaths - ANOVA <https://www.bibmath.net/dico/index.php?action=affiche&quoi=.a/anova.html>