efrei
PARIS PANTHÉON-ASSAS UNIVERSITÉ

RAPPORT CONVEX & OPTIMISATION

Application des techniques d'optimisation à un problème concret de machine learning

Evan Arneau

Doria Deramchi

Mariam Maksimous

Sommaire

I. Contexte et Dataset 3
1. Première partie du projet : 4
2. Deuxième partie du projet : Exploration 5
II. Prédiction utilisée 8
III. Analyse des techniques d'optimisations existantes 11
IV. Optimisation utilisée 11
V. Convexité des fonctions à optimiser pour entraîner un modèle 15

I. Contexte et Dataset

Les canaux de réservation d'hôtels en ligne ont radicalement changé les possibilités de réservation et le comportement des clients. Un nombre important de réservations d'hôtel sont annulées pour cause d'annulation ou de non-présentation. Les raisons typiques d'une annulation sont un changement de plan, un conflit d'horaire, etc. L'annulation est souvent facilitée par la possibilité de le faire gratuitement ou, de préférence, à faible coût, ce qui est bénéfique pour les clients de l'hôtel, mais constitue un facteur moins souhaitable et susceptible de réduire les revenus pour les hôtels.

Nous voulons pouvoir prédire si le client va honorer sa réservation ou l'annuler.

Notre dataset est composé de 36275 lignes et 19 colonnes. Ses différents attributs contiennent les détails de réservation des clients. Comme détaillé ci-dessous :

Booking_ID : identifiant unique de chaque réservation.

no_of_adults: Nombre d'adultes.

no_of_children: Nombre d'enfants.

no_of_weekend_nights : Nombre de nuits de week-end (samedi ou dimanche) que le client a passé ou réservé à l'hôtel.

no_of_week_nights : Nombre de nuits en semaine (du lundi au vendredi) pendant lesquelles le client a séjourné ou réservé son séjour à l'hôtel.

type_of_meal_plan : Type de plan de repas réservé par le client.

required_car_parking_space : Le client a-t-il besoin d'une place de parking ? (0 - Non, 1- Oui).

room_type_reserved : Type de chambre réservée par le client. Les valeurs sont chiffrées (encodées) par INN Hotels.

lead_time : Nombre de jours entre la date de réservation et la date d'arrivée.

arrival_year : Année de la date d'arrivée.

arrival_month : Mois de la date d'arrivée.

arrival_date : Date du mois.

market_segment_type : Désignation du segment de marché.

repeated_guest : Le client est-il un habitué ? (0 - Non, 1- Oui)

no_of_previous_cancellations : Nombre de réservations précédentes qui ont été annulées par le client avant la réservation actuelle.

no_of_previous_bookings_not_canceled : Nombre de réservations précédentes qui n'ont pas été annulées par le client avant la réservation en cours.

avg_price_per_room : Prix moyen par jour de la réservation ; les prix des chambres sont dynamiques. (En euros)

no_of_special_requests : Nombre total de demandes spéciales formulées par le client (par exemple, étage élevé, vue depuis la chambre, etc.)

booking_status : Drapeau indiquant si la réservation a été annulée ou non.

```
df = pd.read_csv("Hotel Reservations.csv")
df
```

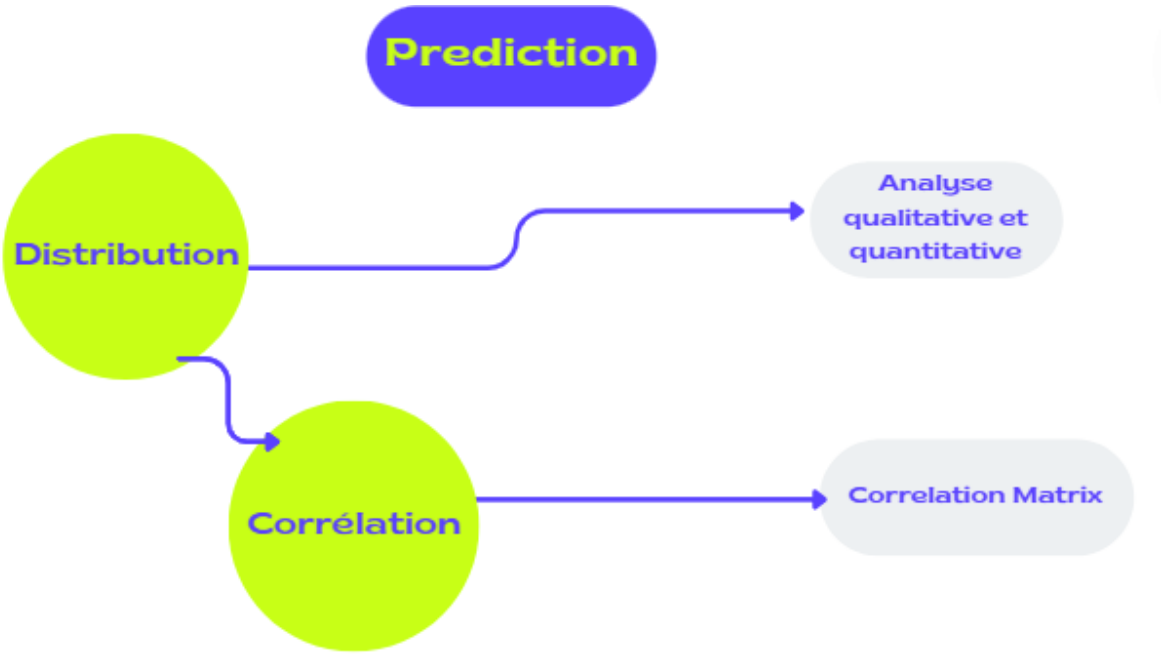
	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_res
0	INN00001	2	0	1	2	Meal Plan 1	0	Room_
1	INN00002	2	0	2	3	Not Selected	0	Room_
2	INN00003	1	0	2	1	Meal Plan 1	0	Room_
3	INN00004	2	0	0	2	Meal Plan 1	0	Room_
4	INN00005	2	0	1	1	Not Selected	0	Room_
...	...	...	...	...	...	...	...	
36270	INN36271	3	0	2	6	Meal Plan 1	0	Room_
36271	INN36272	2	0	1	3	Meal Plan 1	0	Room_
36272	INN36273	2	0	2	6	Meal Plan 1	0	Room_
36273	INN36274	2	0	0	3	Not Selected	0	Room_
36274	INN36275	2	0	1	2	Meal Plan 1	0	Room_

36275 rows × 19 columns

1. Première partie du projet :

La première partie du projet s'est concentrée sur la compréhension des informations de base sur les données, telles que leur taille, les informations sur les colonnes, les lignes vides, les doublons et les formats.

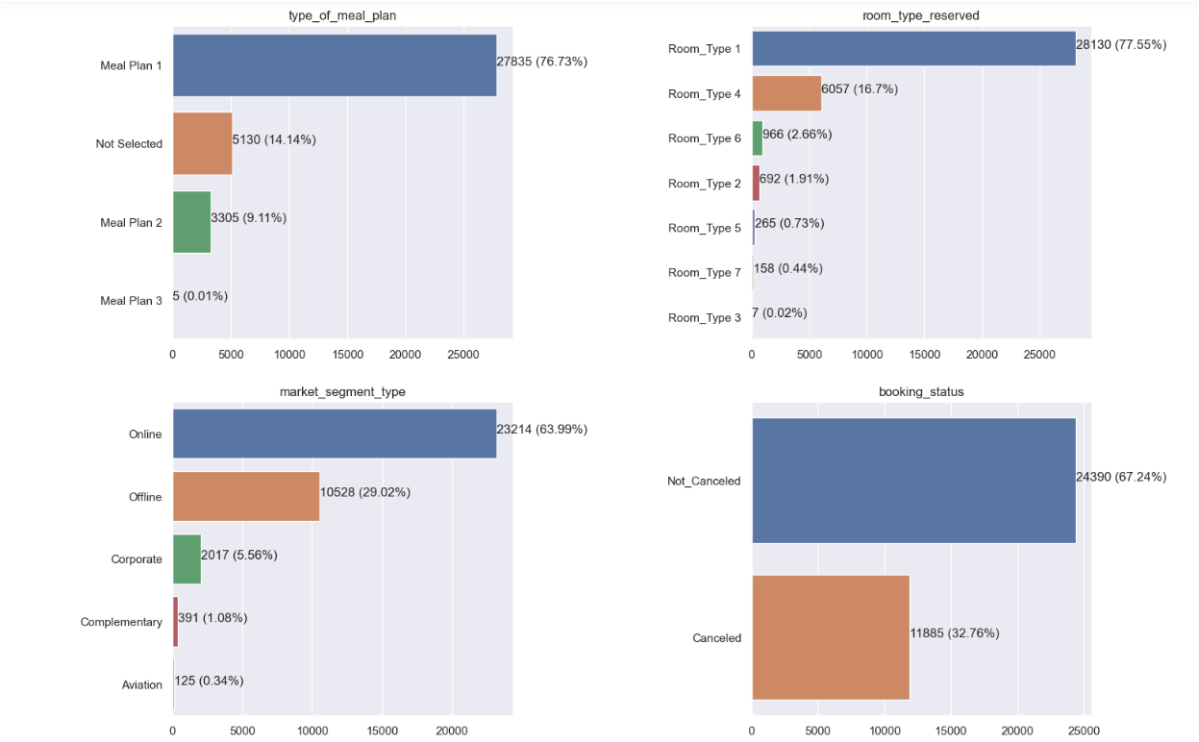
2. Deuxième partie du projet : Exploration



Tout d'abord, nous voulions réaliser une analyse qualitative des colonnes de notre dataset ayant des valeurs objet (des strings ici). Pour ce faire, nous disposions de 5 colonnes objet. Néanmoins, pour cette analyse nous avons ignoré la colonne Booking_ID puisque celle-ci ne nous fournissait aucune information pertinente quant à l'impact sur l'annulation des réservations d'hôtel.

Nous avons ainsi codé une fonction qualitative_analysis. Pour cela, nous avons réalisé une matrice 2x2 pour afficher chacune de nos 4 colonnes (nous avons 4 pos (1,1), (1,2), (2,1), (2,2))) et sélectionné la caractéristique (feature) que nous voulions analyser et ainsi compté le nombre d'éléments par type de valeur.

Enfin, un calcul du pourcentage par valeur a été effectué et affiché près de sa valeur correspondante (cf. figure ci-dessous).



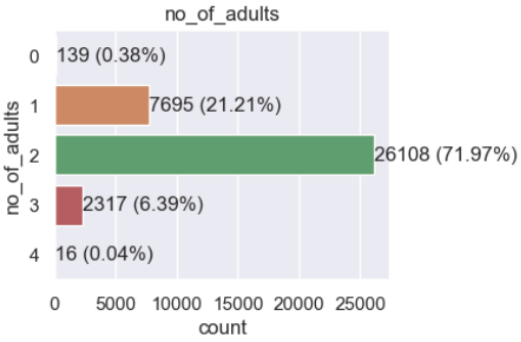
Nous pouvons ainsi constater que dans 76,73% des cas, les clients choisissent le type de repas numéro1, que dans 77,55% des cas, le type de chambre réservé était la room_type_1. Ou encore que 67,24% des réservations n'étaient pas annulées par les clients.

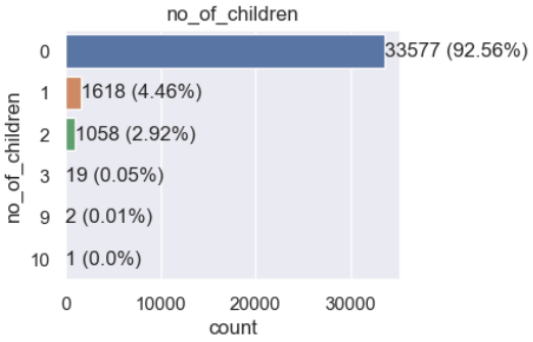
En effet, nous remarquons que le pourcentage d'annulation des réservations est de 32,76%. Un pourcentage non négligeable qui a pour conséquence une perte de revenus considérable pour ces hôtels.

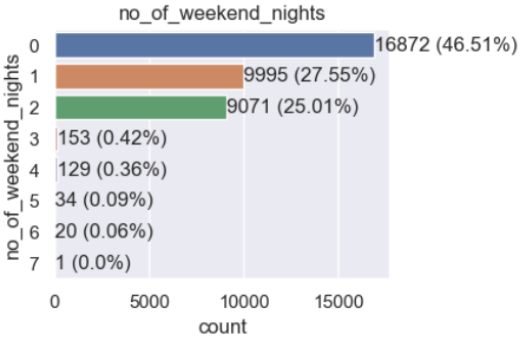
Analyse quantitative :

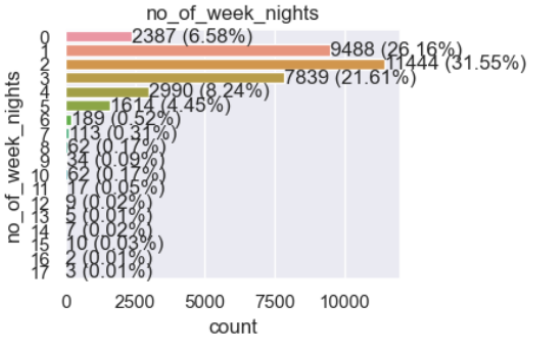
Suite à cela, nous avons mené une analyse quantitative de notre dataset.

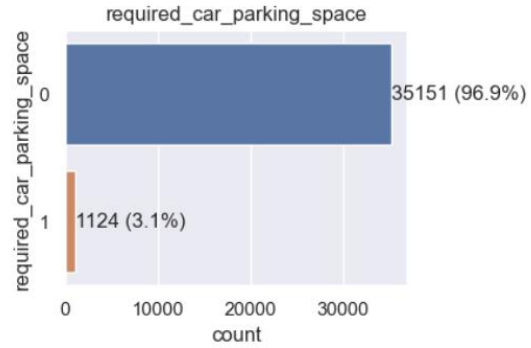
À titre d'exemple, il était à noter que la majorité des réservations de chambre concernaient deux adultes (71,97 %), que 96,9 % des clients avaient besoin d'un espace de stationnement et que 97,44 % d'entre eux étaient des habitués.

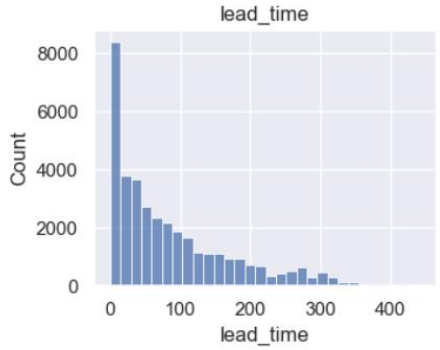

no_of_adults
no_of_adults
0
1
2
3
4
139 (0.38%)
7695 (21.21%)
26108 (71.97%)
2317 (6.39%)
16 (0.04%)
0
5000
10000
15000
20000
25000
count

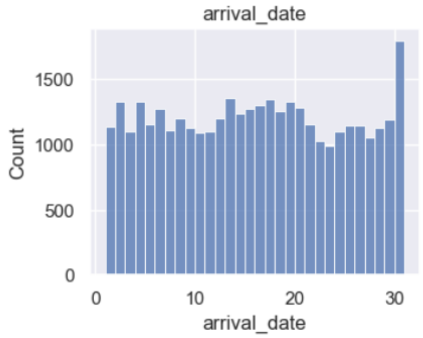

no_of_children
no_of_children
0
1
2
3
9
10
33577 (92.56%)
1618 (4.46%)
1058 (2.92%)
19 (0.05%)
2 (0.01%)
1 (0.0%)
0
10000
20000
30000
count

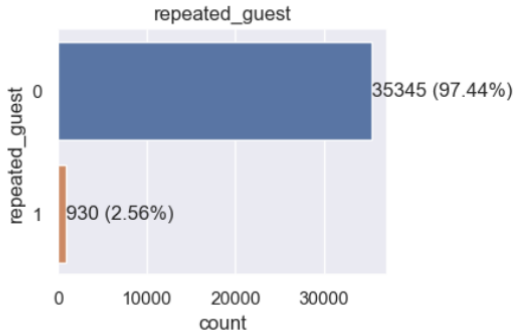

no_of_weekend_nights
no_of_weekend_nights
0
1
2
3
4
5
6
7
16872 (46.51%)
9995 (27.55%)
9071 (25.01%)
153 (0.42%)
129 (0.36%)
34 (0.09%)
20 (0.06%)
1 (0.0%)
0
5000
10000
15000
count


no_of_week_nights
no_of_week_nights
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
2387 (6.58%)
9488 (26.16%)
11444 (31.55%)
7839 (21.61%)
2990 (8.24%)
1614 (4.45%)
189 (0.52%)
113 (0.31%)
62 (0.17%)
34 (0.09%)
62 (0.17%)
17 (0.05%)
9 (0.02%)
5 (0.01%)
7 (0.02%)
10 (0.03%)
2 (0.01%)
3 (0.01%)
0
2500
5000
7500
10000
count

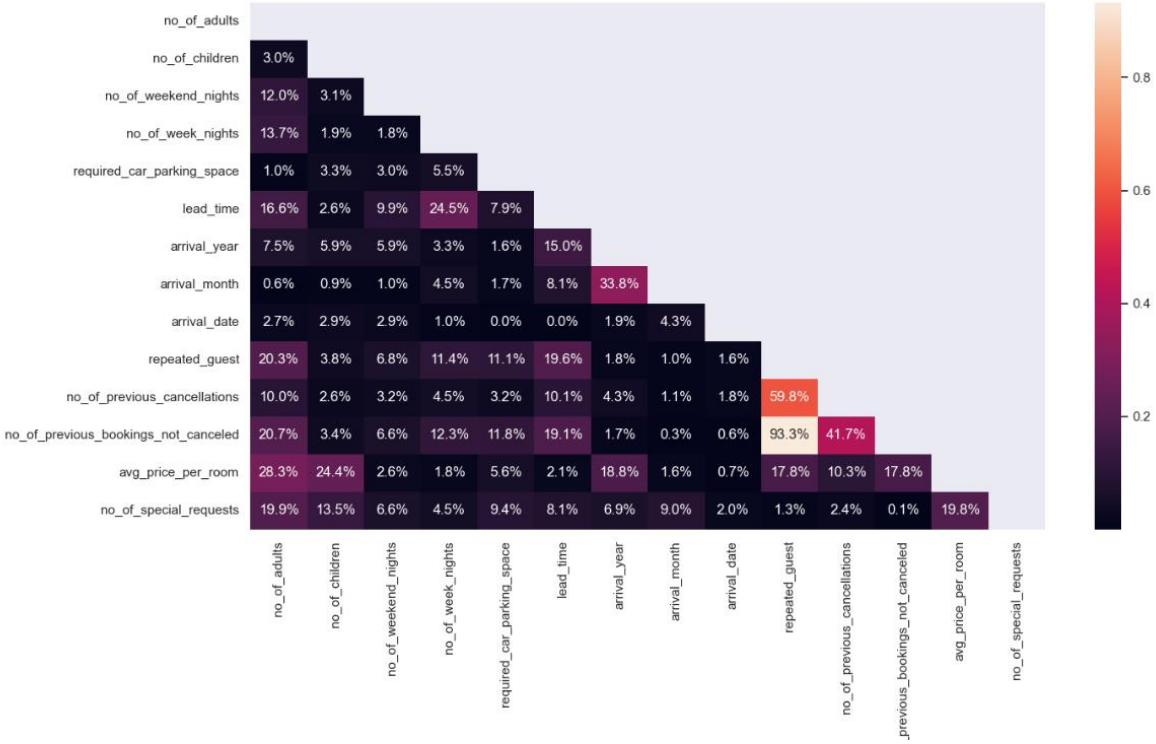

required_car_parking_space
required_car_parking_space
0
1
35151 (96.9%)
1124 (3.1%)
0
10000
20000
30000
count


lead_time
Count
8000
6000
4000
2000
0
0
100
200
300
400
lead_time

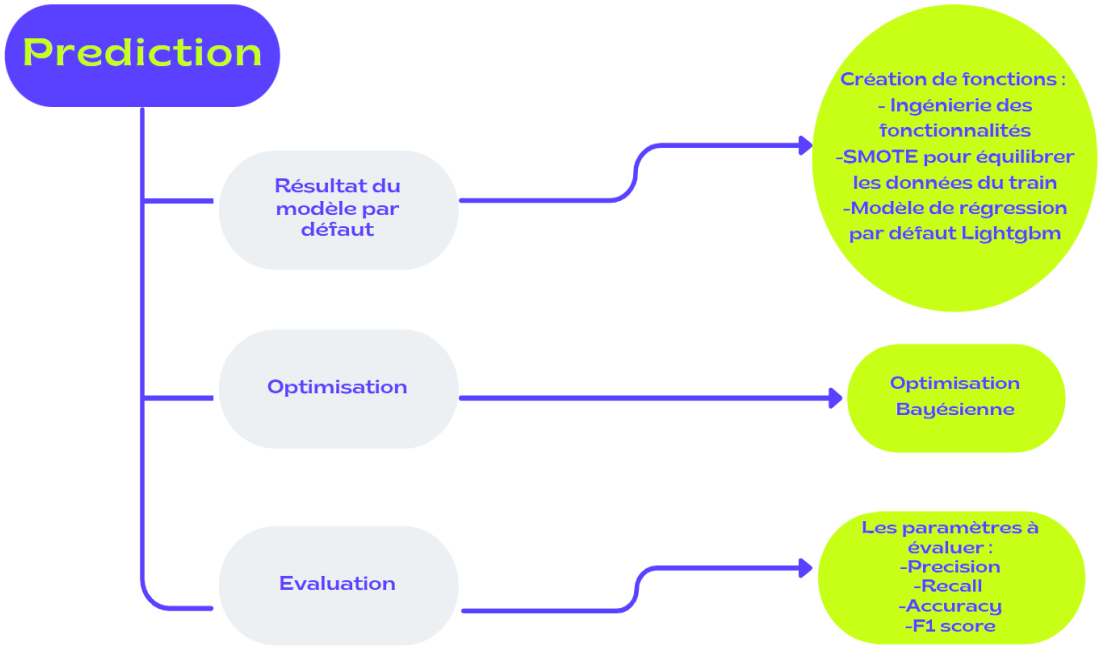

arrival_date
Count
1500
1000
500
0
0
10
20
30
arrival_date


repeated_guest
repeated_guest
0
1
35345 (97.44%)
930 (2.56%)
0
10000
20000
30000
count

Matrice de corrélation :



II. Prédiction utilisée



Le modèle de prédiction utilisé dans la fonction model () est un LightGBMClassifier. Il s'agit d'un algorithme de gradient boosting qui est souvent utilisé pour résoudre des problèmes de classification binaire ou multiclasse.

Dans ce cas-ci, il est utilisé pour prédire si une réservation sera effectuée ou non, en raison de sa haute performance, de sa précision, de sa robustesse aux données manquantes et de sa capacité à éviter le surapprentissage. Avant l'entraînement, des techniques de prétraitement des données ont été appliquées pour éliminer tout biais et équilibrer les données. Ceci a été fait pour garantir que les données d'entraînement soient représentatives et que le modèle puisse apprendre des caractéristiques significatives pour une prédiction précise. Le modèle est entraîné sur les données transformées et équilibrées avec SMOTETomek afin de prédire si une réservation sera effectuée ou non (booking_status est la variable cible). La fonction retourne également les mesures de performance du modèle telles que la précision, le rappel et le F1 score pour chaque classe.

La fonction treatment() effectue le prétraitement des données. Elle commence par effectuer une copie de l'ensemble de données d'entrée (dte). Ensuite, elle applique une transformation de puissance (np.log()) sur les variables lead_time et avg_price_per_room. Cette transformation est utile pour réduire l'effet des valeurs extrêmes et améliorer la distribution des données. Ensuite, elle applique un encodage de label (LabelEncoder()) sur toutes les variables catégorielles. Enfin, elle sépare les variables explicatives (X) de la variable cible (y).

La fonction unbalanced() utilise la technique SMOTE (Synthetic Minority Over-sampling Technique) pour équilibrer les données. Cette technique consiste à créer des données synthétiques pour les classes minoritaires (dans ce cas, les réservations). Cela permet d'équilibrer les classes et d'améliorer les performances du modèle.

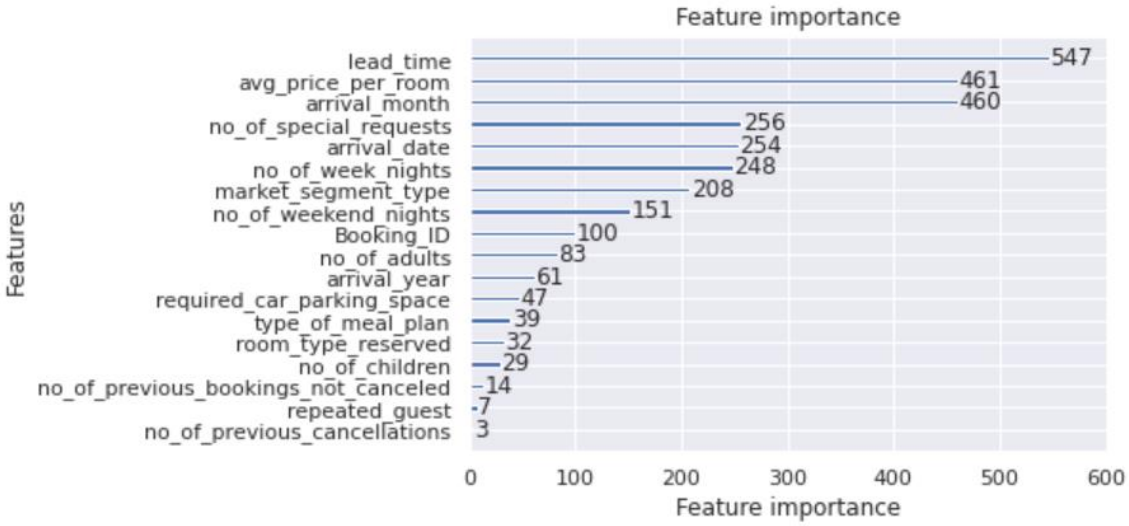
La fonction model() effectue l'apprentissage et l'évaluation du modèle. Elle utilise la technique de validation croisée stratifiée (StratifiedKFold) avec 10 plis pour diviser les données en ensemble d'entraînement et ensemble de test. Pour chaque pli, elle applique la fonction unbalanced() pour équilibrer les classes dans l'ensemble d'entraînement. Ensuite, elle entraîne un modèle LightGBM (LGBMClassifier()) sur l'ensemble d'entraînement et évalue ses performances sur l'ensemble de test à l'aide de la mesure de classification (précision, rappel, f-score) et de la précision globale (accuracy).

Enfin, la fonction model () retourne les performances moyennes du modèle sur les 10 plis, ainsi que l'importance des variables explicatives selon le modèle LightGBM (lg.plot_importance(model)).

Voici, les résultats obtenus :

Accuracy : 88.14887744707785

	Precision	Recall	F1_score
0	83.428117	79.739581	81.502193
1	90.348048	92.246822	91.278885



La métrique recall :

$$recall = \frac{TP}{TP+FN}$$

Pour rappel, le recall ou true positive rate ou encore hit rate (taux de détection), correspond au taux d'individus positifs détectés par le modèle :

Il mesure la capacité du modèle à détecter l'ensemble des individus positifs.

On constate que le "recall 0" est le plus faible. L'objectif est donc de l'améliorer, ce qui signifie que nous devons identifier le plus grand nombre possible de personnes qui annuleront leur réservation.

Si notre modèle a un recall élevé, cela signifie que l'on peut être confiant dans sa capacité à prédire l'annulation d'une réservation par un client, lorsque cela se produira effectivement. De même, si le modèle indique qu'un client ne va pas annuler sa réservation, nous pouvons également avoir confiance en cette prédiction, car il prédira qu'un client annulera sa réservation seulement s'il le fera réellement.

III. Analyse des techniques d'optimisations existantes

Pour optimiser les performances d'un modèle de machine learning, il existe plusieurs approches d'optimisation qui peuvent être utilisées, telles que :

L'optimisation par grille (Grid Search) : qui consiste à tester systématiquement toutes les combinaisons d'hyperparamètres spécifiées dans une grille prédéfinie pour trouver les meilleures performances du modèle.

L'optimisation aléatoire (Random Search) : qui consiste à générer aléatoirement un ensemble d'hyperparamètres à tester pour le modèle et sélectionner ceux qui donnent les meilleures performances.

L'optimisation bayésienne : qui est une méthode itérative qui utilise des modèles probabilistes et une fonction d'acquisition pour explorer efficacement l'espace des hyperparamètres du modèle.

Les algorithmes évolutifs : qui sont des techniques d'optimisation basées sur la sélection naturelle et l'évolution biologique pour rechercher les meilleures performances du modèle.

Dans tous les cas, l'objectif est de trouver la combinaison optimale d'hyperparamètres pour le modèle qui donne les meilleures performances sur les données de test ou de validation.

IV. Optimisation utilisée

Dans notre cas d'annulation de réservation d'hôtel, l'optimisation bayésienne était pertinente car elle nous a permis d'explorer efficacement l'espace des

hyperparamètres, en utilisant les résultats précédents pour orienter la recherche vers les régions les plus prometteuses de l'espace.

Pour orienter notre choix vers l'optimisation bayésienne, nous avons également considéré que cette méthode est particulièrement adaptée pour les problèmes de modélisation où les fonctions coût sont complexes et difficiles à évaluer.

En utilisant cette méthode d'optimisation, nous avons pu explorer plus efficacement l'espace des hyperparamètres en évaluant seulement un petit nombre de configurations, ce qui nous a permis d'obtenir de meilleurs résultats avec un temps de calcul plus court. Cela était particulièrement utile dans notre cas de prédiction d'annulation de réservation d'hôtel, où nous avions une grande quantité de données et un grand nombre d'hyperparamètres à régler pour obtenir les meilleurs résultats possibles.

L'**optimisation bayésienne** est une méthode qui permet de trouver les valeurs optimales d'un ensemble d'hyperparamètres pour un modèle donné en minimisant une fonction de coût (ou d'erreur). Cette méthode est itérative et utilise une approche probabiliste pour construire un modèle de prédiction de la fonction de coût.

La méthode commence par définir une fonction objectif à optimiser, souvent une fonction de coût comme l'erreur de validation croisée ou la précision du modèle. L'algorithme va ensuite construire un modèle de prédiction pour cette fonction objectif, en utilisant une distribution de probabilité pour chaque hyperparamètre. Cette distribution permet de générer des points à évaluer pour déterminer quelle est la meilleure configuration d'hyperparamètres.

Ensuite, l'algorithme évalue la fonction objectif pour chaque point généré et utilise cette information pour mettre à jour le modèle de prédiction. Il peut ainsi prédire quelle sera la meilleure configuration d'hyperparamètres pour minimiser la fonction de coût.

L'optimisation bayésienne est pertinente dans notre cas car elle permet de trouver les valeurs optimales des hyperparamètres pour un modèle LightGBM qui sera utilisé pour prédire si un client annulera ou non sa réservation d'hôtel. En utilisant cette méthode, nous avons pu explorer efficacement l'espace des hyperparamètres, qui est souvent très grand, et trouver rapidement les valeurs qui permettent d'obtenir les meilleures performances du modèle en termes de précision, de rappel, de F1-score ou d'autres

métriques d'évaluation. Cela nous nous a permis de maximiser l'efficacité de notre modèle et d'améliorer ses performances.

Explication code :

La fonction hyperparameter(params) définit un processus d'ajustement des hyperparamètres utilisant la validation croisée pour trouver les valeurs optimales des hyperparamètres du modèle LightGBM précédemment implémenté. La fonction prend en charge une liste d'hyperparamètres (params), qui comprend le learning rate, colsample_bytree, max_depth, et le nombre d'estimateurs (trees).

La fonction prépare ensuite les données pour l'évaluation en appelant une fonction appelée "treatment" sur un dataframe (df), qui renvoie la matrice des caractéristiques (X) et le vecteur cible (y).

La fonction définit ensuite un dictionnaire d'hyperparamètres en utilisant les valeurs de la liste params et crée un classificateur LightGBM (mdl) en utilisant ces hyperparamètres et un état aléatoire fixe (0).

Ensuite, cette dernière utilise la validation croisée 10 fois pour évaluer la précision du classificateur sur les données, en utilisant la moyenne négative du score de précision comme métrique d'évaluation. La fonction imprime ensuite les hyperparamètres et le score de précision moyen négatif.

Enfin, la fonction renvoie le score de précision moyen négatif.

---

gp_minimize est une fonction de la bibliothèque Scikit-Optimize qui implémente une optimisation bayésienne pour trouver les hyperparamètres optimaux d'un modèle.

L'optimisation bayésienne est une méthode d'optimisation itérative qui utilise une combinaison de modèles probabilistes et d'une fonction d'acquisition pour explorer efficacement l'espace des hyperparamètres. À chaque étape, elle ajuste un modèle sur les résultats de la fonction objective évaluée aux points précédemment explorés, puis utilise ce modèle pour sélectionner le prochain point d'évaluation.

Dans le cas de cette implémentation, la fonction gp_minimize est utilisée pour trouver les hyperparamètres optimaux pour un modèle LightGBM en minimisant la valeur retournée par la fonction hyperparameter qui calcule la moyenne de la précision de la classification de la validation croisée négative sur 10 plis. L'optimisation est effectuée

en utilisant les bornes d'hyperparamètres spécifiées dans la liste space_gb et en effectuant un total de 50 appels de fonction, dont 10 appels initiaux aléatoires.

```
 ------------------------------
Iteration No: 49 ended. Search finished for the next optimal point.
Time taken: 57.2236
Function value obtained: -0.8894
Current minimum: -0.8904
Iteration No: 50 started. Searching for the next optimal point.
[0.1, 0.99, 26, 2000] -0.8879675023991629

 ------------------------------
Iteration No: 50 ended. Search finished for the next optimal point.
Time taken: 59.0226
Function value obtained: -0.8880
Current minimum: -0.8904
```

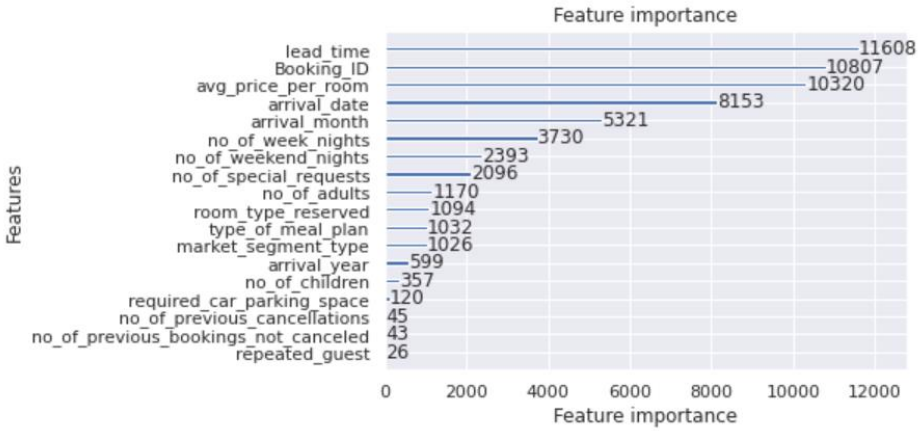
La fonction "model_2" utilise une stratégie de validation croisée pour évaluer la performance du modèle LightGBM avec les hyperparamètres optimaux trouvés par la fonction "hyperparameter". Le modèle est entraîné sur les données d'entraînement et évalué sur les données de test pour chaque fold de la validation croisée. La fonction retourne la précision, le rappel et le score F1 pour les classes positives et négatives.

La stratégie pour optimiser le recall 0 consiste à maximiser la précision, le rappel et le score F1 pour la classe négative (0), tout en minimisant la même métrique pour la classe positive (1). Cela signifie que nous voulons un modèle qui prédise correctement la classe négative autant que possible, tout en évitant de faire des prédictions fausses positives pour la classe positive.

Résultats obtenus après optimisation :

Accuracy : 88.2150432761273

	Precision	Recall	F1_score
0	81.288816	83.980858	82.386498
1	92.111463	90.278803	91.127476



Le recall est passé d'environ 79,73 % à 83,98%. Cela signifie que le modèle prédit mieux les annulations de réservations (Recall 0) après l'optimisation bayésienne, tout en maintenant un niveau de précision (Precision 0) élevé.

Un "recall 0" élevé signifie que le modèle est capable de prédire avec confiance qu'un client va annuler sa réservation si cela se produit effectivement, et donc d'identifier un maximum de vrais positifs. Par conséquent, l'amélioration du "recall 0" après l'optimisation bayésienne signifie que le modèle est maintenant plus précis pour prédire les annulations de réservation, ce qui est positif dans notre cas.

V. Convexité des fonctions à optimiser pour entraîner un modèle

Il est courant que les fonctions de perte ne soient pas convexes dans l'apprentissage automatique, ce qui signifie qu'il peut y avoir plusieurs minima locaux et que la fonction de perte peut avoir des crêtes et des vallées multiples.

Cela peut affecter les modèles de plusieurs façons. Tout d'abord, si un algorithme d'optimisation traditionnel est utilisé pour entraîner le modèle, il peut rester bloqué dans un minimum local suboptimal et ne pas atteindre la solution globale optimale. De plus, si l'algorithme d'optimisation n'est pas conçu pour gérer les crêtes et les vallées multiples, il peut être très lent et coûteux en calcul.

Pour résoudre ce problème, plusieurs approches peuvent être envisagées. L'une consiste à utiliser des algorithmes d'optimisation plus sophistiqués, comme l'optimisation bayésienne ou les descentes de gradient stochastiques, qui sont mieux équipés pour gérer les fonctions non convexes. Dans notre cas, nous avons choisi d'utiliser l'optimisation bayésienne pour optimiser les hyperparamètres de notre modèle de prédiction d'annulation de réservation d'hôtel.

Une autre approche consiste à modifier la fonction de perte elle-même pour la rendre plus facilement optimisable. Cela peut être accompli en utilisant des méthodes telles que la régularisation, qui ajoute des termes de pénalité à la fonction de perte pour encourager la simplicité du modèle, ou la normalisation, qui met à l'échelle les données pour les ramener à une plage de valeurs plus petite.

Enfin, il est important de noter que dans certains cas, la non-convexité peut être bénéfique pour l'apprentissage automatique. Par exemple, les réseaux de neurones profonds sont souvent non convexes et peuvent trouver des représentations de données très utiles. Cependant, il est important de comprendre comment la non-convexité peut affecter le modèle et de choisir l'algorithme d'optimisation approprié en fonction des besoins spécifiques de l'application.