Introduction

Dans le cadre du cours *Apprentissage supervisé et Data challenge*, il nous a été demandé de participer à une *compétition* sur *Kaggle*, visant à tester ce qu'on a appris en cours. Ce rapport résume ce qu'on a pu implémenter: les différents modèles utilisés, les différentes approches pour tester leurs *efficacité*, le traitement des données et la méthode de sélection du meilleur modèle.

Le *challenge* est structuré en deux problèmes d'apprentissage supervisé:

- Un problème de *classification*: prédire, suite à une réservation d'une chambre dans un hôtel, si le client concerné *viendra*, *ne va pas se présenter* le jour *j*, ou *annulera*-t-il sa réservation.
- Un problème de régression, *prédire la popularité* d'un son, sur *spotify*.

Dans ce qui va suivre, on s'intéresse au premier problème, le problème de *classification*. On premier lieu, on s'intéressera aux données, afin d'y extraire de l'information qui nous sera utile pour entraîner des modèles. Puis, on s'intéressera à la procédure utilisée pour le choix des modèles d'apprentissage et puis on commentera sur certaines difficultés rencontrées durant le *data challenge*.

Les données:

Pour le problème de classification, on nous a fourni des données tabulaires ayant des données *numériques* et des données *catégorielles*.

Une première remarque est de voir que les *classes* - (0 pour *check-out* présent), (1 pour *cancel* annulé) et (2 pour *no-show* ne s'est pas présenté) - ne sont pas équilibrées. Il y a une sur-représentation de la classe 0 - plus que la classe 1, qui est plus représentée que la classe 2.

Classes	Proportion
Check-out	62.9409 %
Cancel	36.0664 %
No-show	0.9928 %

Table 1: La proportion des classes dans le jeu de données fourni

Ce que ce tableau nous fait comprendre est que dans la plupart des cas, les clients honorent leurs réservations - dans ce jeu de données. Ceci dit, le manque d'exemples - en proportion - sur *no-show* rend le jeu de données biaisais, une information qui sera prise en compte dans la suite de notre projet.

En se restreignant aux données numériques, on remarque qu'elles ne sont pas corrélées. Ce qui implique, dans un premier temps, qu'on n'a pas d'autre choix que de les garder. Le manque de corrélation fait que chaque colonne peut nous apporter une information supplémentaire sur le problème.

Quant aux données catégorielles, plusieurs choses peuvent être dites. D'abord, plusieurs modèles d'apprentissage ne traitent pas les données catégorielles automatiquement. Ces modèles-là nécessitent un pré-traitement des données. Ce *pre-processing* s'est avéré être très chronophage, et devient un *hyperparamètre* qui doit être optimisé. Dans ce qui va suivre on discutera de quels choix de *pré-processing* qu'on a pris et les motivations derrières ces choix.

Remarques sur les données catégorielles:

Durant le traitement des données catégorielles, on a remarqué que certaines valeurs étaient sur-représentées dans le jeu de données. Un exemple frappant est la colonne *country*. En regardant attentivement cette catégorie, on se rend compte de la chose suivante: *country = 'PRT'* est sur-représentée dans le jeu de données avec une proportion de 40 %. Bien d'autres catégories sont sur-représentées. Certaines ont une explication rationnelle, comme dans le cas de *meal* où *BB* est sur-représenté. C'est tout à fait naturel, étant donné que les réservations d'hôtels sont dans la plupart des cas pour des vacances et donc beaucoup de déplacements - pas de déjeuners dans les hôtels. Évidemment cette sur-représentativité de certaines catégories ne nous donne pas spécialement des informations sur le statuts réservations.

Dans le cas de *meal*, on voit clairement que *BB* est dans tous les statuts de réservations le premier. Ceci dit, en regardant de plus près, on voit bien que la probabilité estimée $\hat{\mathbb{P}}(\text{meal} = \text{'SC'}|\text{reservation-status} = 2) \geq \hat{\mathbb{P}}(\text{meal} = \text{'HB'}|\text{reservation-status} = 2)$ sachant quand-même que la classe *HB* est plus représentée que la classe *SC*.

Problème avec les variables catégorielles:

On remarque qu'on a beaucoup de valeurs possibles pour certaines variables catégorielles. Un *one-hot encoding* serait excessivement encombrant et couteux d'un point de vue dimensionnel et algorithmique. Pour y remedier, une solution possible était de trouver une codification qui puisse à la fois donner une information sur les variables encodées, et qui en plus permet de réduire la dimension de nos données. Le dernier point étant extrêmement important pour la vitesse l'entraînement des modèles. On a décidé d'utiliser plusieurs types d'encodages possibles suivant ce critère: si la colonne a plus ($\geq$) de trois valeurs, l'encoder, sinon faire un *one-hot encoding*.

Codification possible des variables catégorielles:

Une première approche a été de regarder comment codifier des variables catégorielles en gardant le maximum d'information qu'elles nous rapportent sur le *status des réservations* au moyens d'estimations de probabilités conditionnelles.

Soit X une colonne catégorielle qui peut prendre des valeurs $\{x_1, \ldots, x_p\}$ où $p \geq 3$. Si $X[i]$ est la i^{me} valeur dans la colonne X, on estime les probabilités suivantes:

$$\hat{X}_{\text{check\_out}}[i] = \hat{\mathbb{P}}(\text{check\_out}|X = X[i])$$
$$\hat{X}_{\text{cancel}}[i] = \hat{\mathbb{P}}(\text{cancel}|X = X[i])$$
$$\hat{X}_{\text{no\_show}}[i] = \hat{\mathbb{P}}(\text{no\_show}|X = X[i])$$

Et on estime aussi le nombre suivant:

$$\hat{n}[i] = \sum_{j=1}^{N} \mathbf{1}_{X[j]=X[i]}$$

Il suffit alors de remplacer la colonne X par le vecteur de ces probabilités estimées, en ajoutant $\hat{n}$.

Avantages:

- Encodage informatif et supervisé: on capture des relation statistiques entre la variable catégorielle et la variable cible.
- Réduction de dimension: au lieu d'utiliser un *one-hot encoding* qui risque de donner plus de 200 modalités, cet encodage permet de réduire considérablement la dimension, indépendamment du nombres de modalités dans chaque catégorie.
- Une codification interprétable.
- Compatibilité avec plusieurs modèles.

Inconvénients:

- Possibilité de sur-apprentissage: une modalité rare peut impliquer une probabilité de 1.
- Manque de robustesse: une seule observation peut donner un signal fort et faux
- Pas de pondérations sur les catégories rares.
- Dépend énormement du jeu de données.

Pour palier aux inconvénients, on a estimé les probabilités suivantes:

$$\hat{p}_{\text{check\_out}} = \hat{\mathbb{P}}(\text{check\_out})$$
$$\hat{p}_{\text{cancel}} = \hat{\mathbb{P}}(\text{cancel})$$
$$\hat{p}_{\text{no\_show}} = \hat{\mathbb{P}}(\text{no\_show})$$

Et puis on a transformé nos données comme ceci:

$$\hat{X}_{\text{check\_out}}[j] \leftarrow \frac{n[j] \times \hat{X}_{\text{check\_out}}[j] + K \times \hat{p}_{\text{check\_out}}}{\hat{n}[j] + K}$$

$$\hat{X}_{\text{cance}}[j] \leftarrow \frac{n[j] \times \hat{X}_{\text{cancel}}[j] + K \times \hat{p}_{\text{cancel}}}{n[j] + K}$$

$$\hat{X}_{\text{no\_show}}[j] \leftarrow \frac{n[j] \times \hat{X}_{\text{no\_show}}[j] + K \times \hat{p}_{\text{no\_show}}}{\hat{n}[j] + K}$$

où K est un ***hyperparamètre*** de lissage.

Une autre codification fréquentiste possible est de faire le même processus mais cette fois-ci avec les estimateurs suivants:

$$\hat{X}_{\text{check\_out}}[i] = \hat{\mathbb{P}}(X = X[i] | \text{ check\_out})$$
$$\hat{X}_{\text{cancel}}[i] = \hat{\mathbb{P}}(X = X[i] | \text{ cancel})$$
$$\hat{X}_{\text{no\_show}}[i] = \hat{\mathbb{P}}(X = X[i] | \text{ no\_show})$$

avec le même lissage qu'avant. Le tableau suivant représente la grosse différence entre les deux codifications.

Type de codage	**Ce que ça représente**	**Avantage**	**Inconvénient**
$P(y \mid x)$	Influence de la variable sur la cible	Directement utile pour la prédiction	Risque de fuite de cible si mal géré
$P(x \mid y)$	Distribution des catégories pour chaque classe	Peut mieux capturer la structure des classes	Moins directement discriminant pour la prédiction

Table 2: Comparaison entre les deux types de codage probabiliste des variables catégorielles.

Il reste alors les catégories ayant moins ($<$) de 3 valeurs possibles qu'on codifie avec du *one-hot encoding*.

Entraînement:

Première idée:

Ce qu'on a fait, dès lors où on a choisi la codification, est de sélectionner des modèles pour l'entraînement. On en a vu pas mal en cours. On a crée une liste de plusieurs modèles, et on les a entrainé sur 80 % des données *train.csv*, avec une *validation croisée* à 5 blocs.

Pour réduire les biais, à chaque étape de la *cross-validation*, dans le bloc d'entrainement, on calcule la codification mentionnée plus haut, on transforme nos données d'entraînement, ainsi que les données de validation, ***en utilisant uniquement la codification calculée dans le bloc d'entraînement*** avec l'hypothèse $\mathbb{P}_{\text{entraînement}}(Y|X) = \mathbb{P}_{\text{validation}}(Y|X)$.

Maintenant grâce aux blocs de validation, on enregistre les erreurs *OOB* pour chaque model et chaque étape de la *CV*. Avec ces erreurs *OOBs*, on entraîne un *meta-learner*, pour une aggregation d'experts. Pourquoi ? En effet, la raison vient du fait qu'on veuille choisi le meilleur *meta-learner* qui sache mieux combiner entre les ***types*** de modèles, en se basant sur leurs prédictions réelles après des entraînements différents. Ainsi on capture "universellement" les meilleurs combinaisons sur des types de modèles différents. Autrement dit, si une forêt aléatoire entrainée à chaque étape sur différentes données d'entraînement, et qu'elle généralise bien sur l'ensemble de la *CV*, elle aura un coeifficient élevé pour l'aggregation d'experts.

On a remarqué, en implémentant l'idée, que les modèles types fôrets aléatoire avaient plus de capacité à bien généraliser.

Table 3: Résultat de log-loss par modèle et par fold

Modèle	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**
HistGradientBoostingClassifier	0.3124	0.3159	0.3139	0.3167	0.3044
ExtraTrees	0.4331	0.4346	0.4331	0.4440	0.4348
LogisticRegression	0.4244	0.4310	0.4276	0.4329	0.4288
RandomForest	0.3750	0.3759	0.3762	0.3800	0.3735
CatBoost	0.3338	0.3371	0.3386	0.3383	0.3293
Moyenne	**0.3757**	**0.3789**	**0.3779**	**0.3824**	**0.3742**

Meta-Learner	Log-loss	Weighted F1
HistGradientBoostingClassifier	0.2846	0.8711

Table 4: Performance du meta-learner sur les prédictions OOF

Fold	Log-loss	Weighted F1
1	0.3320	0.8478
2	0.3362	0.8485
3	0.3401	0.8465
4	0.3329	0.8505
5	0.3084	0.8656
Mean	0.3299	0.8518
Std	0.0111	0.0070

Table 5: Performance du meta-learner sur les folds de test et statistiques globales

De ce qui a précédé, on peut déduire que les méthodes *boosting* sont plus appropriés pour nos données. On en déduit aussi que l'agrégation d'experts fait mieux en moyenne sur la log-loss que chacun des modèles.

Deuxième idée:

Ici, on va chercher à garder la codification que pour le modèle *HistGradientBoostingClassifier* puisqu'il ne gère pas les données catégorielles de manière *"native"*. Et on utilise les modèles *LightGBM* et *CatBoost* en laissant les données catégorielles originelles.

95 % des données d'entraînement serviront à entraîner les trois modèles. La moitié des 5% restantes, serviront à faire du *online learning* avec l'algorithme *EWA - Exponential Weighted Algorithm* pour l'aggregation d'experts:

Algorithm 1 Exponential Weights Algorithm (EWA) pour classification

Require: Nombre d'experts N, learning rate $\eta > 0$, nombre de rounds T

Initialiser les poids des experts : $w_{i,1} = 1$ pour $i = 1, \ldots, N$
for $t = 1$ **to** T **do**
  Recevoir les prédictions de chaque expert : $\hat{y}_{i,t}$ pour $i = 1, \ldots, N$
  Normaliser les poids : $p_{i,t} = \frac{w_{i,t}}{\sum_{j=1}^{N} w_{j,t}}$
  Agréger les prédictions : $\hat{y}_t = \sum_{i=1}^{N} p_{i,t} \hat{y}_{i,t}$
  Observer la vraie étiquette y_t
  Calculer la perte de chaque expert : $\ell_{i,t} = \ell(y_t, \hat{y}_{i,t})$
  Mettre à jour les poids : $w_{i,t+1} = w_{i,t} \cdot \exp(-\eta \cdot \ell_{i,t})$
end for
return Prédictions agrégées $\hat{y}_1, \ldots, \hat{y}_T$

D'abord le résulatat de l'entraînement et test sur la moitié des 5 % de données de test:

Modèle	**Log-loss**	**F1-weighted**
CatBoost	0.3085	0.8660
LGBM	0.3148	0.8602
HistGradientBoosting	0.3100	0.8620
Méthode agrégée (Test final)	0.2994246203229059	0.8704161752594727

Table 6: Performances des modèles sur la première moitié le jeu de test

On voit que l'aggregation d'experts fait mieux que chacun des modèles sur la première moitié du jeu de données test - 5 %.

On test l'aggregation d'experts sur la seconde moitié des données tests:

Modèle	**Log-loss**	**F1-weighted**
Aggregation EWA	0.30402657019077556	0.8641351183805687

Table 7: Performances de l'aggregation d'experts par *EWA* sur la seconde moitié des données test

Conclusion:

On a réussi à implémenter plusieurs modèles différents d'apprentissages supervisé, on a su gérer les données avec des codifications - qu'on n'a pas vu en cours-, on a implémenté des méthodes d'online learning pour pouvoir entraîner au mieux l'aggrégation d'experts.

Ce qu'on aurait pu faire: essayer de trouver les meilleurs hyperparamètres pour nos codifications de données, les meilleurs hyperparamètres pour les modèles et peut-être voir certaines architectures.