Projet air_passengers

MOSEF 2020

Louis Grunenwald

Paul Leydier

1. Feature Engineering

Nous avons commencé par effectuer du feature engineering sur le champ DateOfDeparture, car les dates font parti des données les plus améliorables. Nous avons ainsi créé les **données temporelles** suivantes (sous forme de booléens) :

- Jours fériés
- Vacances scolaires
- Week-ends
- Proche du week-end

Nous avons ensuite intégré des **données géographiques** au jeu de données :

- Latitude
- Longitude
- Elévation
- Population de la ville principale de l'aéroport

En plus des données relatives aux aéroports et à la temporalités des évènements, nous avons envisagé de rajouter des données démographiques et économiques plus ciblées sur les usagers et leurs Etats. En effet, nous avons établi quelques hypothèses, telles que :

- un salaire élevé permet peut-être de plus voyager.
- Un taux de chômage élevé contribue à diminuer la richesse de l'Etat et donc intrinsèquement les emplois proposés dans celui-ci (Perte d'attractivité).
- Ou encore un taux d'enfant par femme qui, dans le cas où il est élevé, contribuerait à une utilisation plus fréquente de l'avion (réunion de famille, fête, décès,...). Cette variable, couplée au taux de migration domestique peut-être déterminant. Un Etat avec un fort taux de natalité mais une migration forte pourrait avoir un transit assez conséquent.

De ces hypothèses, nous avons extrait 6 variables :

- Unemployment_Rate : Taux de Chômage

- Peer_Capita_Income/Median_Income/Average_Income : salaires par personnes / salaires médian / salaires moyens.
- Net_domestic_migration_rate : le taux domestique de migration pour 1000 personnes
- TFR : Taux de natalité des femmes

Chaque variables est liée à l'Etat de provenance. Exemple pour l'Etat de Floride avec l'aéroport de Miami :

AirPort	City	State	Unemployment_Rate	Peer_capita_income	Median_income	Average Wage	Net_domestic_migration_rate_per_1000_ha	TFR 2018
PHX	Phoenix	Arizona	4,8	25715	50068	90726	11,61	1.79

Enfin, nous avons effectué du **feature engineering** sur les données à notre disposition :

- Nous avons calculé la distance entre les deux aéroports via les latitudes et longitudes intégrées plus tôt
- Nous avons effectué un clip sur les outliers des données numériques, mais cette méthode n'a pas porté de bons résultats, et nous l'avons donc annulée
- Nous avons calculé la durée écoulée depuis le dernier vol similaire (mêmes aéroports de départ et arrivé) selon le jeu de données disponible. Néanmoins, ce temps de calcul s'est avéré trop long pour un jeu de données aussi grand que celui du serveur ramp
- Nous avons calculé le nombre de vols par an sur ce trajet

2. Modélisation

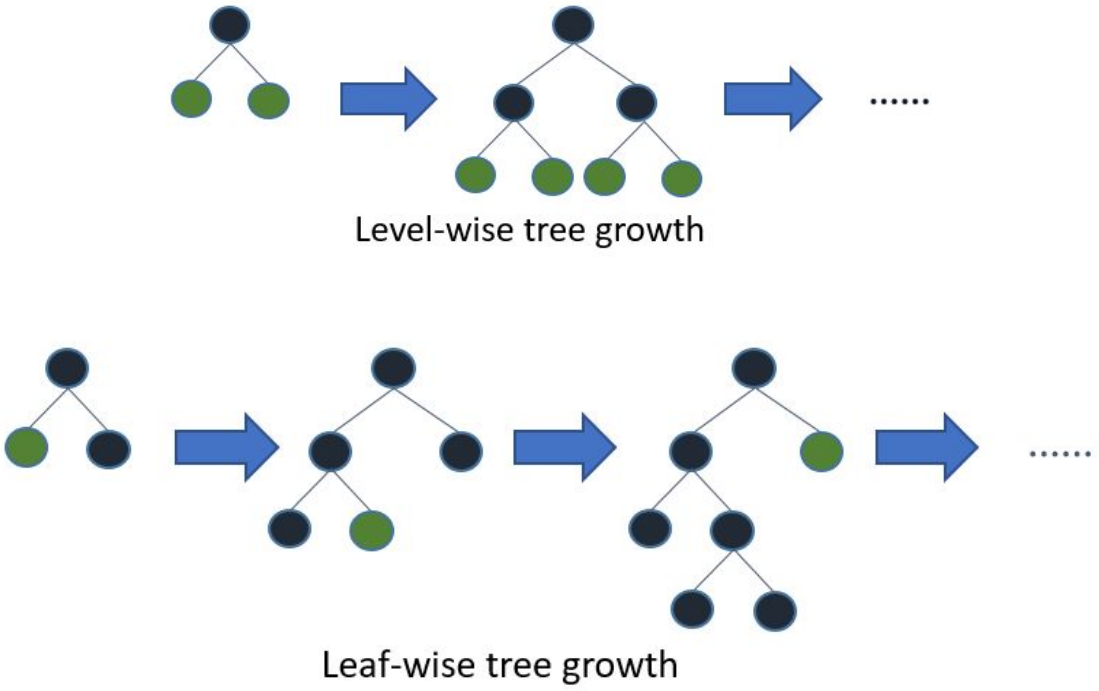
Au vu de la nature du challenge, à savoir obtenir le meilleur sur un métrique défini (à savoir la Root Mean Squared Error) dans un problème de régression classique, sans spécifiquement tenir compte de la variance du modèle, nous avons décidé de nous concentrer sur les algorithmes de Gradient Boosting.

En effet, cette classe d'algorithmes est réputée pour obtenir de très bons résultats même à partir d'un petit jeu de données. Notre jeu de données étant effectivement relativement petit (bien que celui présent sur la plateforme ramp semble être plus grand à en juger par les meilleurs scores obtenus et les bien plus grands temps d'entraînements), nous n'avons pas jugé utile d'explorer les performances de réseaux neuronaux.

2.1. LightGBM

Comme XGBoost et CatBoost, le LightGBM est un algorithme de gradient boosting. Développé par Microsoft, et plus récent que XGBoost, il est réputé pour obtenir des performances similaires voir meilleures en des temps d'entraînements inférieurs,

grâce à sa croissance *Leaf-wise* par opposition à la croissance *Level-wise* du XGBoost.

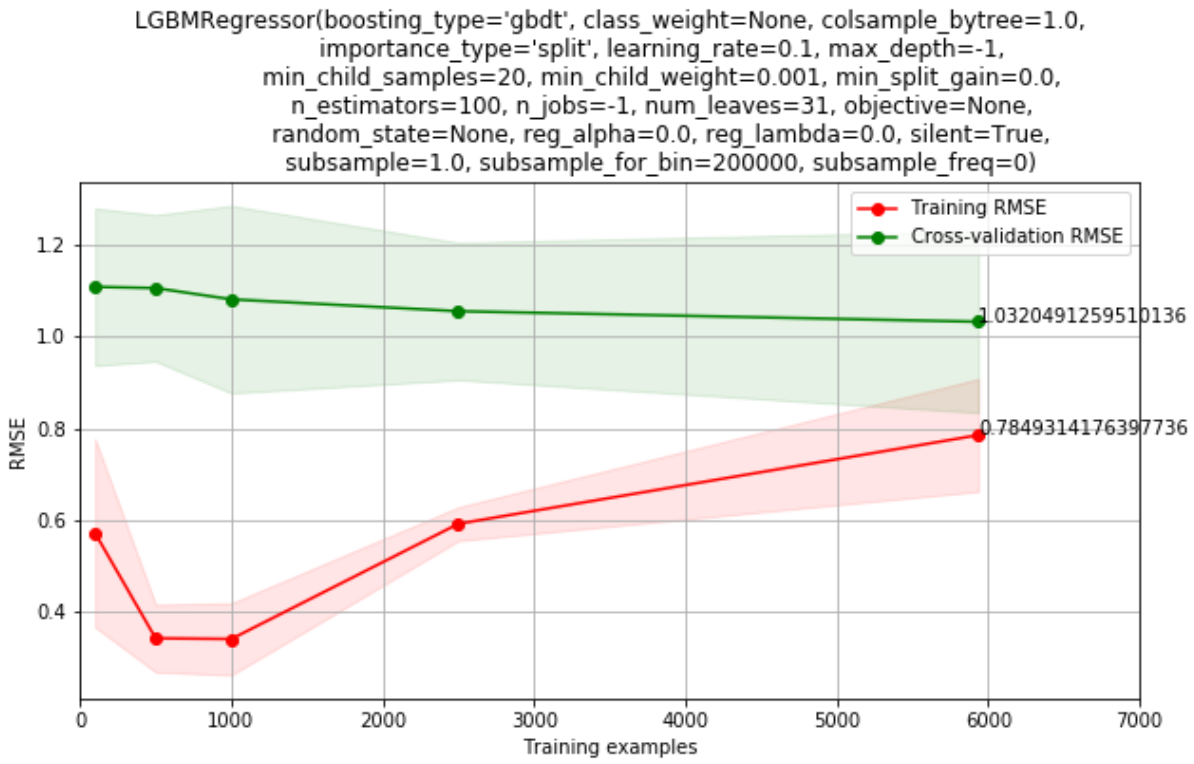


Source: https://lightgbm.readthedocs.io/en/latest/Features.html#leaf-wise-best-first-tree-growth

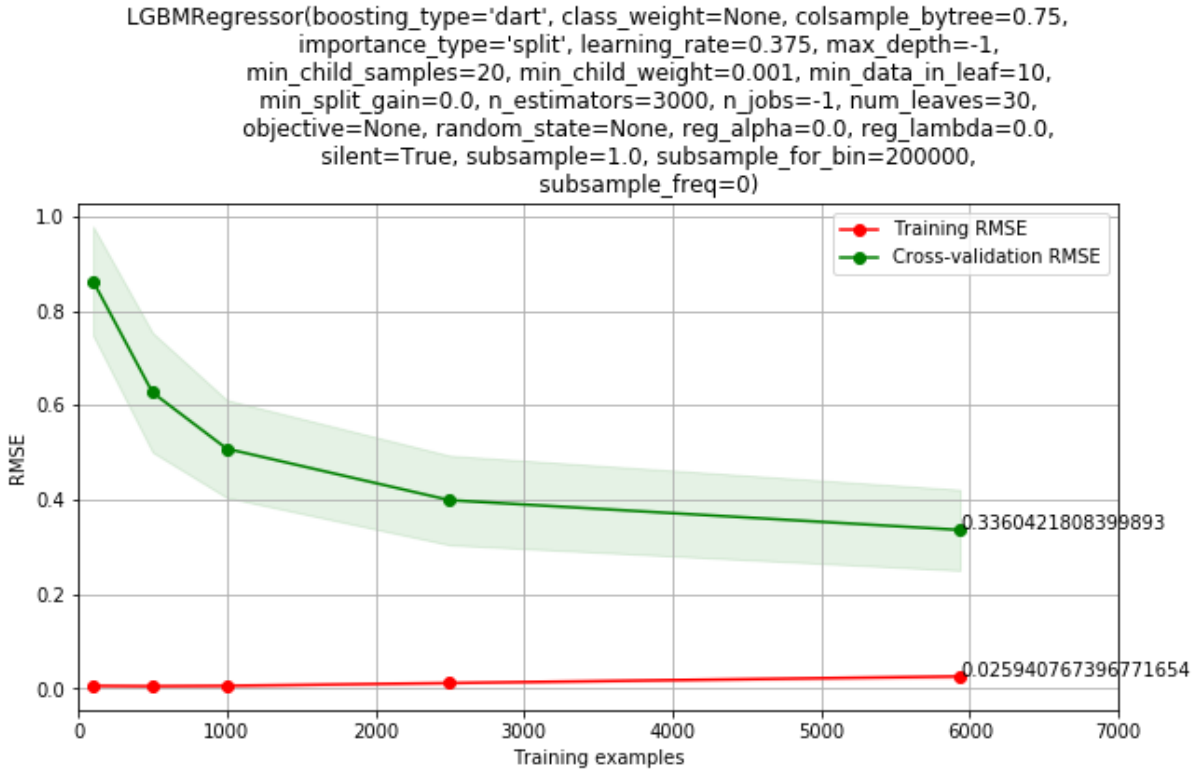
Nous avons donc décidé d'utiliser un LightGBM. Notre méthode d'optimisation des hyperparamètres a été la suivante :

- Etablissement de la liste des hyperparamètres par ordre d'importance, et de leur impact théorique sur le tradeoff biais-variance
- Evaluation d'un LGBM sans hyperparamètres via les RMSE de train / cross-validation / test, mais aussi via la learning curve pour comprendre le tradeoff biais-variance
- GridSearchCV sur les un ou deux hyperparamètres les "plus importants" en veillant à sélectionner des valeurs cohérantes avec le tradeoff biais-variance. Répéter en "zoomant" la grille sur les hyperparamètres les plus performants
- Une fois les paramètres précédents optimisés, répéter la démarche précédente avec un à deux nouveaux hyperparamètres.

Nous avons ainsi pu obtenir les résultats suivants :



Hyperparamètres par défaut



Hyperparamètres optimisés

Nous avons volontairement fait en sorte que :

- Notre modèle overfit le jeu de données d'entraînement local
- La RMSE de cross-validation de notre modèle continue à diminuer avec le nombre d'exemple

En effet, le jeu de données du serveur ramp semble être bien plus grand que le jeu d'entraînement local, et pourra donc profiter d'un modèle plus complexe, qui ne parvient pas à converger sur notre jeu de données limité. Cette stratégie s'est avérée payante, puis notre LightGBM s'est classé premier au score et dispose d'un temps d'entraînement bien plus faible que le second (1 916 secondes contre 5 955 secondes).

team	submission	rmse	train time [s]	validation time [s]	max RAM [MB]	submitted at (UTC)
Orlogskapten	Ne_pas_regarder	0.000	3.167275	0.884649	0.0	2020-03-07 17:49:13
pLeydier	lgbm_10	0.254	1916.633044	292.317576	0.0	2020-03-15 11:07:18
pLeydier	lgbm_9	0.255	2621.605416	472.739697	0.0	2020-03-14 14:11:16
Meerwan007	grossekhapta2	0.259	5955.252410	742.458782	0.0	2020-03-16 15:03:18
enaihali	Projetgrinta2	0.260	4422.596044	563.591398	0.0	2020-03-16 16:06:56
sneibam	LightGBM_max5	0.260	5568.076966	738.430209	0.0	2020-03-07 13:07:56
ginkulsergei	starting_Sergioo	0.262	6209.603623	707.263500	0.0	2020-03-14 18:39:26
sneibam	LightGBM_max4	0.262	4846.983404	653.221668	0.0	2020-03-07 11:17:31
sneibam	LightGBM_max3	0.262	4855.905891	651.408140	0.0	2020-03-07 07:39:22
Meerwan007	grossekhapta4	0.265	2644.102257	87.396048	0.0	2020-03-16 17:34:19

Leaderboard le 17/03 à 9h30.

2.2. CatBoost

Après le LightGBM, nous avons décidé d'utiliser le CatBoost développé par Yandex. Il s'agit d'un algorithme de Boosting tout comme le LightGBM ou le XGB, censé fournir de meilleurs résultats tant dans la modélisation que dans le temps d'exploitation.

<u>Présentation des efficacités de trois algorithmes de Boosting :</u>

	XGBoost	Light BGM		CatBoost	
Parameters Used	max_depth: 50 learning_rate: 0.16 min_child_weight: 1 n_estimators: 200	max_depth: 50 learning_rate: 0.1 num_leaves: 900 n_estimators: 300		depth: 10 learning_rate: 0.15 l2_leaf_reg= 9 iterations: 500 one_hot_max_size = 50	
Training AUC Score	0.999	Without passing indices of categorical features 0.992	Passing indices of categorical features 0.999	Without passing indices of categorical features 0.842	Passing indices of categorical features 0.887
Test AUC Score	0.789	0.785	0.772	0.752	0.816
Training Time	970 secs	153 secs	326 secs	180 secs	390 secs
Prediction Time	184 secs	40 secs	156 secs	2 secs	14 secs
Parameter Tuning Time (for 81 fits, 200 iteration)	500 minutes	200 minutes		120 minutes	

Sources : https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Le Light GBM est globalement meilleur sur des données exclusivement numériques, cependant dans un cas de données mixtes :

	Light GBM	CatBoost
Training Score	0.999	0.887
Test Score	0.772	0.816
Training Time	326 secs	390 secs
Prediction Time	156 secs	14 secs

Score : On peut voir que le LightBoost est nettement supérieur sur l'entraînement mais est très peu robuste contrairement au CatBoost. En effet, tandis que l'AUC train/test reste assez inchangé pour le CatBoost, le LightBoost lui le voit totalement modifié de par son manque de robustesse

Temps :En terme de temps, il n'y a pas de grosses différence, en efft, l'algorithme LightGBM apprend un peu plus vite que le CatBoost tandis que ce dernier prédit lui plus vite (10 fois).

Après ces analyses, nous décidons donc d'utiliser le CAtboost sur notre jeu de données.

Le catBoost dans sa conception ne nous oblige pas à binariser nos modalités catégorielles. En effet, chacun des variables catégorielle peut être laissée sans « one hot encoding » ou autre procesus de binérisation. Nous retravaillons donc la tables et laissons nos variables catégorielles telles quelle, même celle initiale, comme les aéroports d'arrivée et de départ. Procéder de cette façon nous donne un avantage, réduire la dimensionnalité de nos données. Le problème de dimension est un enjeu primordial en Data Science, trop de données, amène beaucoup de bruits et ce bruit peut nuire au résultat du modèle. A la vue du nombre d'observation limitée que nous avons à disposition, il serait par conséquent intéressant de travailler avec ce CatBoost.

Implémentation du CatBoost :

Tout comme le LIghtBoost, le CatBoost possède beaucoup de paramètres permettant de l'adapter aux données sur lesquelles l'on souhaite travailler.

Il existe trois grandes méthodes d'optimisation des paramètres, le GridSearch, le RandomSearch et le BayesianSearch. Pour optimiser les données, nous choisissons d'utiliser l'optimisation Bayésienne.

Optimisation Bayésienne : Il s'agit d'une optimisation basée sur une approche probabiliste. L'optimisation bayésienne fonctionne en construisant une distribution postérieure des fonctions (par un processus gaussien) qui décrit le mieux la fonction que nous souhaitons optimiser. À mesure que le nombre d'observations augmente, la distribution postérieure s'améliore et l'algorithme cible mieux l'intervalle sur lesquels les paramètres doivent évoluer afin de déterminer les paramètres optimaux.

Nous optimisons par rapport à la metrique de la RMSE. Nous appliquons un coefficient de -1 à chacune des valeurs de RMSE sortie pour choisir la valeur négative maximum (ce qui nous donne le minimum positif).

Résultats :

C'est ces paramètres que nous utiliserons dans la modélisation sur le serveur RAMP.

Au long de ce déploiement du modèle CatBoost, nous n'avons malheureusement pas réussi à modifier le fichier regressor.py pour y ajouter une liste de variable catégorielle, nous avons donc continué avec des variables catégorielle binarisées, ce qui tronque la plus-value du CatBoost.

Ainsi donc le résultat final du CatBoost n'est pas meilleur que celui du LightGBM. Nous pensons cependant que , si nous avions réussi à changer le fichier regressor de telle sorte à spécifier nos variables catégorielle, nous aurions eu un meilleur résultat que le LightBoost.