Justine Lopes-Vicente

Compte rendu d'amélioration d'un modèle de classification

J'ai commencé par créer un csv à partir des tableaux numpy pour éviter de recréer les valeurs à chaque fois.
Ensuite j'importe ce csv dans un jupyter notebook pour faire mes tests.
Le score de départ à améliorer est un accuracy de 0.788.

Piste d'amélioration de départ :

- feature selection
- tester plusieurs modèles
- régler les hyperparamètres

1) Sélection des features :

Utilisation de **SelectFromModel** de la librairie sklearn, qui permet de choisir les meilleures features.
Voici les features sélectionnées : Index(['5', '6', '7', '8', '9', '10', '12', '14', '15', '20', '22', '26', '27', '28', '35', '38', '46', '47', '50', '52', '63']
et voici le résultat non concluant : 0.6322418136020151
Les résultats n'étant vraiment pas bons, je préfère partir dans une autre direction pour le moment pour peut-être y revenir après avoir fait plus de veille sur ce sujet si j'ai le temps.

2) Test de différents modèles de classification courants et comparaison des scores :

	Model	Accuracy_training	Accuracy_testing
0	SVC	0.798865	0.788413
1	KNN	0.889029	0.783375
2	RandomForestClassifier	1.000000	0.886650
3	GaussianNB	0.663304	0.612091
4	LogisticRegression	0.772383	0.738035
5	DecisionTreeClassifier	1.000000	0.755668
6	LinearDiscriminantAnalysis	0.796974	0.788413
7	QuadraticDiscriminantAnalysis	0.837327	0.790932

Avec ce tableau on peut voir les scores de base (sans amélioration) de plusieurs modèles de classifications. J'ai choisi celui qui donnait le meilleur score pour l'améliorer : le Random Forest Classifier.

3) Test de régler les hyperparamètres du meilleur modèle Random Forest Classifier avec GirdSearchCV :

Meilleurs hyperparamètres : {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 200}
Meilleure performance (score de validation croisée) : 0.9041644346566674
Précision entraînement: 1.0, précision test: 0.8790931989924433

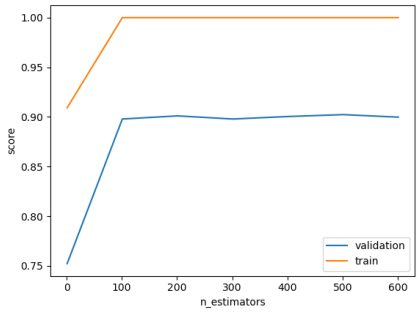
Le GirdSearchCV, n'a pas vraiment amélioré mon modèle, il est difficile de trouver les paramètres à tester du premier coup. car la limite de GridSearchCV est que si on choisit trop de paramètres il sera très long à exécuter.

4) Test de régler les hyperparamètres du meilleur modèle Random Forest Classifier avec BayesSearchCV :

Meilleurs hyperparamètres : OrderedDict([('max_depth', 10), ('min_samples_leaf', 0.1), ('min_samples_split', 0.1), ('n_estimators', 174)])
Précision entraînement: 0.6986128625472888, précision test: 0.6498740554156172

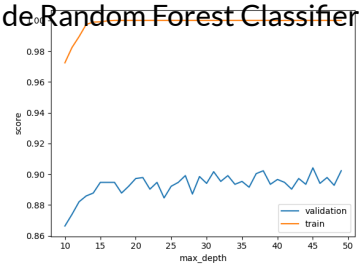
Cette méthode non plus n'a pas donné de bon résultats, je dois mal définir les paramètres de départ.

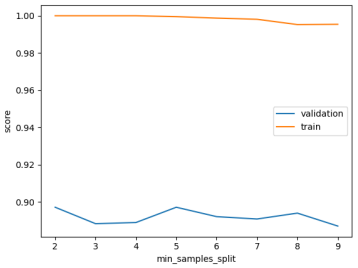
5) Test de régler les hyperparamètres un par un dans un range donné avec une **validation_curve** pour une meilleur visualisation des hyperparamètres pour mieux cibler les valeurs de départ dans le **GirdSearchCV** :

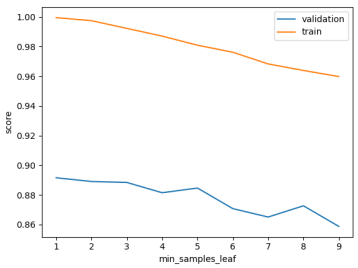


Voici la validation_curve qui permet de visualiser les scores en fonction de la valeur du paramètre n_estimators, par contre on ne peut régler qu'un paramètre à la fois. peut être utile au début pour avoir une idée du range à tester ensuite sur GridSearchCV.

Les courbes de validations pour d'autres paramètres de Random Forest Classifier :







Toutes les courbes ne sont pas simples à interpréter, comme pour le paramètre max_depth où le score de validation monte et descend de manière quasi 'cyclique', ce n'est pas facile de choisir la meilleur valeur. Grâce au GridSearchCV on va pouvoir en tester plusieurs, simultanément.

Le GirdSearchCV me donne les meilleurs paramètres pour mon test :
'criterion': 'entropy', 'max_depth': 35, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100

Le score de mon modèle augmente un peu : 0.8942065491183879.
On peut voir que les courbes de validations m'ont aidées à mieux cerner les hyperparamètres et m'ont aidé à choisir des valeurs pour le GirdSearchCV.

6) Test d'une autre librairie BayesianOptimization :

Les tests avec BayesianOptimization n'augmentent pas le score du modèle.
Je pense arriver vers un plafond des possibilités d'optimisation des hyperparamètres pour ce jeux de données avec ce modèle.

A cette étape, je suis à un score de 0.8942065491183879, avec le modèle de RandomForestClassifier et les paramètres suivant : {'criterion': 'entropy', 'max_depth': 35, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

7) Test de la librairie Autogluon

Maintenant j'ai envie de partir dans une autre direction et de tester la librairie **Autogluon** et plus précisément **autogluon.tabular** qui est utilisé pour traiter des données tabulaires.
Il faut juste des entrées nos données et définir les labels, pas besoin de traiter les données.

	model	score_test	score_val	pred_time_test
0	WeightedEnsemble_L2	0.991427	0.957179	0.394604
1	LightGBMXT	0.987393	0.937028	0.030129
2	CatBoost	0.986384	0.931990	0.007090
3	LightGBM	0.986384	0.931990	0.043016
4	XGBoost	0.984367	0.921914	0.021566
5	LightGBMLarge	0.983863	0.919395	0.033380
6	RandomForestGini	0.982350	0.911839	0.073926
7	RandomForestEntr	0.981846	0.909320	0.069448
8	ExtraTreesEntr	0.981341	0.906801	0.063441
9	ExtraTreesGini	0.979324	0.896725	0.069318
10	NeuralNetTorch	0.978820	0.919395	0.055351
11	KNeighborsDist	0.966717	0.833753	0.095411
12	KNeighborsUnif	0.855270	0.831234	0.111462

Voici les résultats ressortis de l'autoML.
Je retrouve les valeurs des hyperparamètres qu'il a utilisé pour RandomForestGini et RandomForestEntr :

({'n_estimators': 300,
'max_leaf_nodes': 15000,
'n_jobs': -1,
'random_state': 0,
'bootstrap': True,
'criterion': 'gini'},)

La seule chose qui change entre les deux c'est le criterion : gini ou entropy.
Et je reteste ces valeurs avec mes données :

Précision entraînement: 1.0, précision test: 0.9045226130653267
Les deux (gini et entropy) donnent la même valeur. Avec Autogluon j'ai pu augmenter un peu le score de mon modèle.
Le weightedEnsemble_L2 est le meilleur modèle dans ce test, mais je ne vais pas l'utiliser, je préfère continuer à augmenter mon modèle initiale.

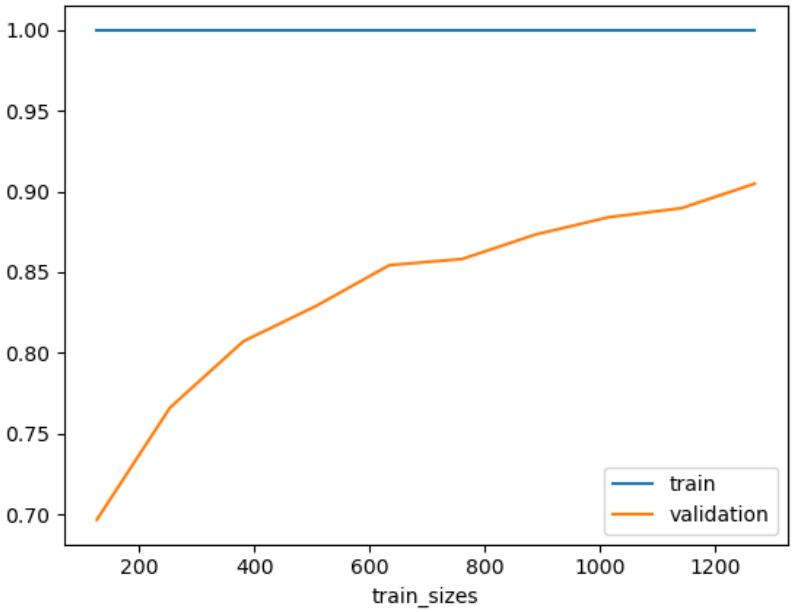
8) Test modifier les données d'entrées : passer de 0.2 à 0.4 secondes

Sophana s'est rendu compte qu'en prenant des morceaux plus grands (0,4 secondes au lieu de 0,2 secondes) on pouvait améliorer les scores. J'ai créé un nouveau csv à partir des morceaux de 0,4 secondes et j'ai testé mon meilleur modèle sur ces données :

RFC = RandomForestClassifier(max_leaf_nodes = 15000, n_jobs = -1, random_state=20,bootstrap=True, n_estimators= 300, criterion='gini')

Précision entraînement: 1.0, précision test: 0.9292929292929293
En effet, cette modification à amélioré mon modèle.

9) Faire une learning curve



La learning curve permet de voir l'évolution des scores en fonction de la taille des données d'entraînement. Ici on peut voir que la courbe des scores de validation n'a pas atteint un plateau, ce qui veut dire qu'avec plus de données on pourrait potentiellement obtenir de meilleurs scores.
On pourrait créer de nouvelles valeurs à partir de celles qu'on a déjà en les modifiant un peu.

10) Améliorations restantes à faire pour encore augmenter le score :

- Augmenter le nombre de données
- Faire de la feature selection