NOTE METHODOLOGIQUE

Projet 7 Parcours Data Scientist
Implémenter un modèle de scoring

https://github.com/colple/Implementez_un_modele_de_scoring

Préambule indispensable dans la compréhension du cheminement de l'obtention du jeu de données avec la gestion du déséquilibre des classes

Mon mentor travaillant exactement dans ce domaine m'a guidée dans le nettoyage des données et le feature engineering nécessaires à l'élaboration du modèle de scoring. De ce fait, le kernel de Kaggle n'a pas été utilisé et les principaux changements sont : (i) la sélection des variables pertinentes pour la création d'un modèle de scoring, (ii) un seul type d'agrégation pour les variables (à l'exception de l'AMT_CREDIT des tables bureau.csv et previous_application.csv), (iii) l'imputation des données manquantes par des valeurs fictives, réalisée dans le métier (-1 pour les variables catégorielles et -2 pour les variables numériques), (iv) la réduction du nombre de données par feature engineering pour certaines variables catégorielles et (v) encodage manuel pour la plupart d'entre-elles. Par la suite, certaines variables ont été éliminées dues à de trop fortes corrélations.

Afin de sélectionner les variables les plus pertinentes, une RFE-CV ("Recursive Feature Elimination with Cross-Validation" ou Élimination Récursive des Caractéristiques avec Validation Croisée) a été réalisée sur 75000 clients à l'aide Random Forest Classifier avec les paramètres de base, exception faite pour le paramètre class_weight. En effet, pour la prise en compte de déséquilibre des classes, un class_weight = 'balanced' a été fixé. En se basant sur la méthode du coude, le point d'inflexion était situé aux alentours de 20 variables. Ainsi, par précaution, 30 variables ont été sélectionnées.

1. Gestion du déséquilibre des classes

Etant en présence d'un jeu de données totalement déséquilibré (~92% de la classe 0 et ~8% de la classe 1), une méthode permettant de gérer le déséquilibre des classes était indispensable. En effet, le déséquilibre des classes pose problème car les algorithmes d'apprentissage automatique ont tendance à privilégier la classe majoritaire, entraînant de mauvaises performances pour la classe minoritaire.

Pour gérer ce déséquilibre, 4 techniques ont été testées sur un Random Forest Classifier optimisé: (i) SMOTE, (ii) oversampling (sur-échantillonnage de la classe minoritaire), (iii) undersampling (sous-échantillonnage de la classe majoritaire) et (iv) attribution de poids différents entre les 2 classes. La métrique d'évaluation pour le choix a été principalement basée sur notre métrique métier (cf page suivante), mais aussi sur le sur-apprentissage (overfitting) et d'autres métriques comme l'AUC ("Area Under the Curve" ou Aire sous la courbe). Cette dernière est couramment utilisée en Machine Learning afin d'évaluer la performance des modèles de classification, en plus particulièrement dans des contextes où les classes sont déséquilibrées.

1.1. SMOTE ("Synthetic Minority Over-sampling Technique"), oversampling et undersampling

SMOTE génère des échantillons synthétiques (*ie* pas de duplication d'échantillons) dans la classe minoritaire en se basant sur les k plus proches voisins, aboutissant au même nombre d'échantillons dans les 2 classes. L**'oversampling** augmente le nombre d'observations de la classe minoritaire afin d'arriver à un ratio classe minoritaire / classe majoritaire satisfaisant, alors que l'**undersampling** diminue le nombre d'observations de la classe majoritaire afin d'arriver au ratio classe minoritaire / classe majoritaire satisfaisant.

Concernant les résultats obtenus, parmi ces 3 méthodes, la technique la plus prometteuse aurait été l'undersampling, suivie de l'oversampling. Néanmoins, ne souhaitant pas perdre de données (ou en générer), une attribution de différents poids pour la classe minoritaire a été testée.

1.2. L'attribution des différents poids

Six différentes attributions de poids pour la classe minoritaire ont été testées (2, 4, 6, 8, 10, 12), avec les meilleures performances avec un class_weights de {0 :1,1 :2}. Ainsi, cette stratégie a été retenue pour la gestion du déséquilibre des classes.

2. La fonction coût métier et les métriques d'évaluation

2.1. La fonction coût métier

La société 'Prêt à dépenser' souhaite minimiser le risque d'octroyer un crédit à des clients pouvant rencontrer des difficultés de paiements, ce qui est préjudiciable en termes de coût. Ainsi, l'établissement d'un coût métier est indispensable.

Terminologie pour la compréhension du coût métier

- FP (False Positive) : Les clients prédits comme présentant un risque alors qu'ils sont en mesure de rembourser leur prêt (perte financière pour la société).
- FN (False Negative) : Les clients prédits comme étant en capacité de rembourser leur prêt, alors qu'ils ne le sont pas (encore plus préjudiciable pour la société).
- TP (True Positive) : Les clients incapables de rembourser leur prêt et prédits de la sorte.
- TN (True Negative) : Les clients capables de rembourser leur prêt et prédits de la sorte.

Comme suggéré, le postulat de base sera que le coût d'un FN est dix fois supérieur au coût d'un FP, donnant ainsi le score métier suivant :

$$\frac{10 * FN + FP}{TP + TN + FP + 10 * FN}$$

2.2. Les métriques d'évaluation

Concernant l'évaluation du modèle, cette dernière n'a pas été réalisée sur une métrique unique mais plutôt sur un ensemble de métriques : (i) le coût métier, (ii) l'AUC et (iii) la matrice de confusion seuillée pour chaque modèle.

En effet, le seuil de base pour la matrice de confusion est fixé à 0,5. De ce fait, un seuil personnalisé a été réalisé pour chaque algorithme testé. Ce seuil est basé sur le meilleur score obtenu.

Outre la prise en compte de ces 3 métriques, le sur-apprentissage (overfitting) a également été considéré pour le choix du modèle, ce dernier pouvant avoir des conséquences sur les prédictions.

Malgré la gestion du déséquilibre des classes, aucun modèle n'a permis une classification parfaite, ce qui était attendue. En effet, le meilleur AUC ne pouvait pas dépasser 0,82, score obtenu par le vainqueur de Kaggle.

3. Méthodologie pour l'obtention du modèle

Après avoir déterminé la façon de gérer le déséquilibre des classes, 4 (voire 5 algorithmes) ont été entraînés : (i) la LogisticRegression, (ii) le CatBoostClassifier, (iii) le LGBMClassifier, (iv) le XGBoostClassifier et (v) le DummyClassifier servant de baseline.

Concernant la séparation des données en jeu d'entraînement et de test, 80% des données ont été utilisées pour le train et 20% pour le test avec une stratification sur y. De plus, les variables numériques présentant de grosses différences de magnitude, ces dernières ont été normalisées *via* le MinMaxScaler de la librairie scikit-learn.

3.1. Entraînement des modèles sans recherche d'hyperparamètres

Dans un premier temps, les différents modèles ont été entraînés avec les paramètres de base exceptions faites pour la gestion du déséquilibre des classes et le Random State, ce dernier étant fixé à 42 afin de s'assurer de la reproductibilité des résultats à chaque lancement du notebook. De plus, concernant la régression logistique, le maximum d'itérations a dû être fixé à 500 afin d'être en mesure de converger.

	DummyClassifier	LogisticRegression	CatBoostClassifier	LGBMClassifier	XGBClassifier
Training Time	0.0081	8.1704	28.9202	1.2504	18.7926
Train Accuracy	91.9271	82.4433	84.1115	82.7786	83.5587
Test Accuracy	91.9272	82.5553	81.3456	82.1683	80.8676
Train AUC	0.5	0.6999	0.8652	0.8054	0.8676
Test AUC	0.5	0.7029	0.7677	0.7675	0.7616
Train Recall	0.0	0.3619	0.6885	0.5617	0.6985
Test Recall	0.0	0.3563	0.5223	0.5096	0.5156
Train Precision	0.0	0.1906	0.2936	0.2489	0.287
Test Precision	0.0	0.1902	0.2217	0.2287	0.2147
Train f1	0.0	0.2497	0.4117	0.345	0.4069
Test f1	0.0	0.248	0.3113	0.3157	0.3032
Best threshold	0.0808	0.2222	0.2121	0.2222	0.2121
Train best score	0.4676	0.4367	0.3141	0.3721	0.3146
Test best score	0.4676	0.4375	0.3961	0.3942	0.4018

Concernant notre métrique métier d'intérêt (best score), de bons résultats sont déjà obtenus pour les 3 derniers algorithmes.

3.2. Entrainement des modèles avec recherche des meilleurs paramètres *via* GridSearchCV

Dans un deuxième temps, une tentative d'optimisation des modèles a été réalisée *via* GridSearchCV. Deux modèles ont été éliminés : (i) La régression logistique, modèle peu performant et (ii) le XGBMClassifier, modèle présentant le plus d'overfitting et très coûteux niveau temps pour l'optimisation des paramètres.

	DummyClassifier	LogisticRegression	CatBoostClassifier	LGBMClassifier	XGBClassifier
Training Time	0.0082	23.9292	8.6766	1.3275	57.8648
Train Accuracy	91.9271	82.2002	82.1933	80.5299	85.4415
Test Accuracy	91.9272	82.5196	81.1115	79.9294	81.131
Train AUC	0.5	0.7048	0.8172	0.805	0.9247
Test AUC	0.5	0.7073	0.7669	0.7679	0.7672
Train Recall	0.0	0.3745	0.6056	0.6066	0.8255
Test Recall	0.0	0.3738	0.5321	0.5519	0.5299
Train Precision	0.0	0.1917	0.2506	0.2311	0.3363
Test Precision	0.0	0.1954	0.2213	0.2131	0.2211
Train f1	0.0	0.2536	0.3545	0.3347	0.4779
Test f1	0.0	0.2567	0.3126	0.3075	0.312
Best threshold	0.0808	0.2222	0.2121	0.202	0.202
Train best score	0.4676	0.4348	0.3611	0.3737	0.2417
Test best score	0.4676	0.4328	0.3947	0.397	0.3952

Ainsi, un fine tuning a été effectué sur les 2 algorithmes restants : (i) le CatBoostClassifier et (ii) le LGBMClassifier présentant de moins bons résultats après cette recherche d'optimisation.

3.3. Fine tuning des 2 modèles prometteurs

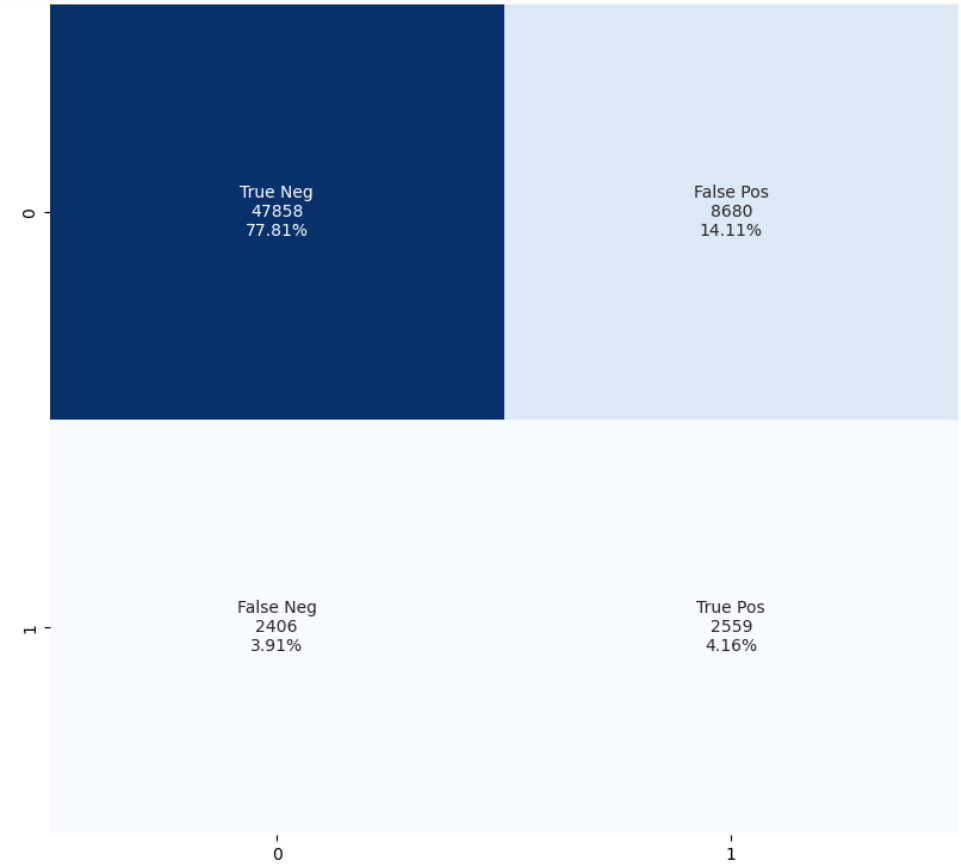
Ce dernier a amoindri les performances du CatBoostClassifier et n'a pas permis d'obtenir de meilleurs résultats que le LGBMClassifier de base. Afin d'évaluer la performance retenue pour la gestion du déséquilibre des classes, cette dernière a été comparée à l'auto_class_weight = 'Balanced' et le class_weight = 'balanced' respectivement pour le CatBoostClassifier et le LGBMClassifier. Il s'est avéré que la gestion du déséquilibre des classes était de moindre qualité avec les paramètres de base (cf tableau récapitulatif ci-dessous).

Au regard de l'ensemble des résultats, le modèle retenu pour la suite du projet a été le **LGMBClassifier avec le class_weight = {0 :1,1 :2} et un RandomState fixé à 42** présentant le meilleur score métier sur le jeu de test.

TABLEAU RECAPITULATIF DES 2 ALGORITHMES D'INTERET

	CatBoost Base auto_class_weight	CatBoost Base avec class_weight retenu	CatBoost 1ère opt	CatBoost 2ème opt	LGBMC Base class_weight	LGBMC Baseavec class_weight retenu	LGBMC 1ère opt	LGBMC 2ème opt
Training Time	69.8722	70.6693	20.8735	70.2175	4.6839	3.4623	5.2072	15.2404
Train Accuracy	81.1421	84.1115	82.1933	82.755	80.801	82.7786	80.5299	83.8078
Test Accuracy	78.4482	81.3456	81.1115	80.3262	80.4042	82.1683	79.9294	80.4107
Train AUC	0.8857	0.8652	0.8172	0.8607	0.8032	0.8054	0.805	0.8967
Test AUC	0.7641	0.7677	0.7669	0.7688	0.7674	0.7675	0.7679	0.7679
Train Recall	0.7899	0.6885	0.6056	0.7024	0.5872	0.5617	0.6066	0.7774
Test Recall	0.5813	0.5223	0.5321	0.5432	0.5482	0.5096	0.5519	0.539
Train Precision	0.2709	0.2936	0.2506	0.2764	0.23	0.2489	0.2311	0.3036
Test Precision	0.2052	0.2217	0.2213	0.2153	0.2172	0.2287	0.2131	0.2152
Train f1	0.4034	0.4117	0.3545	0.3967	0.3306	0.345	0.3347	0.4367
Test f1	0.3034	0.3113	0.3126	0.3083	0.3112	0.3157	0.3075	0.3076
Best threshold	0.5354	0.2121	0.2121	0.202	0.596	0.2222	0.202	0.202
Train best score	0.296	0.3141	0.3611	0.3196	0.3784	0.3721	0.3737	0.2786
Test best score	0.3985	0.3961	0.3947	0.3969	0.3947	0.3942	0.397	0.3977

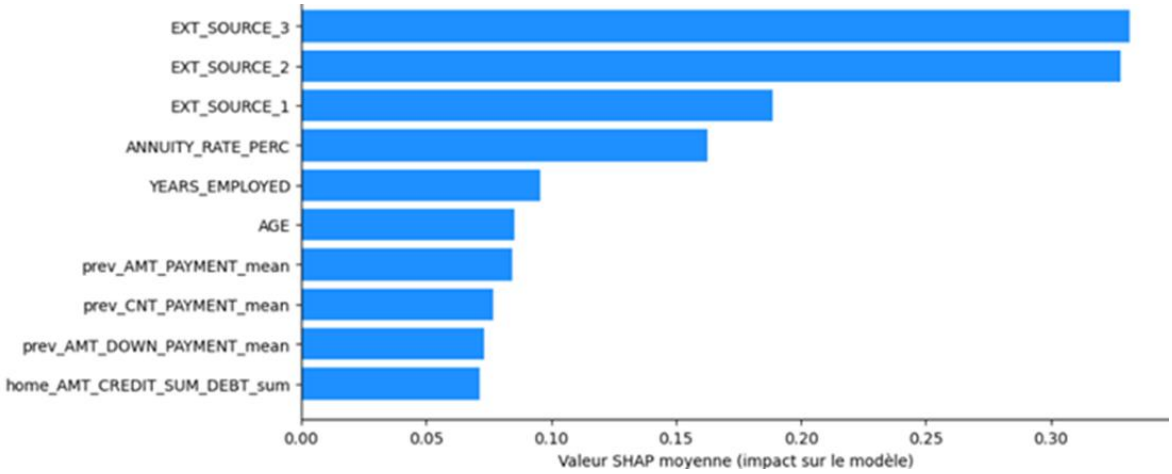
LA MATRICE DE CONFUSION



4. L'interprétabilité globale et locale du modèle : Les valeurs de Shapley

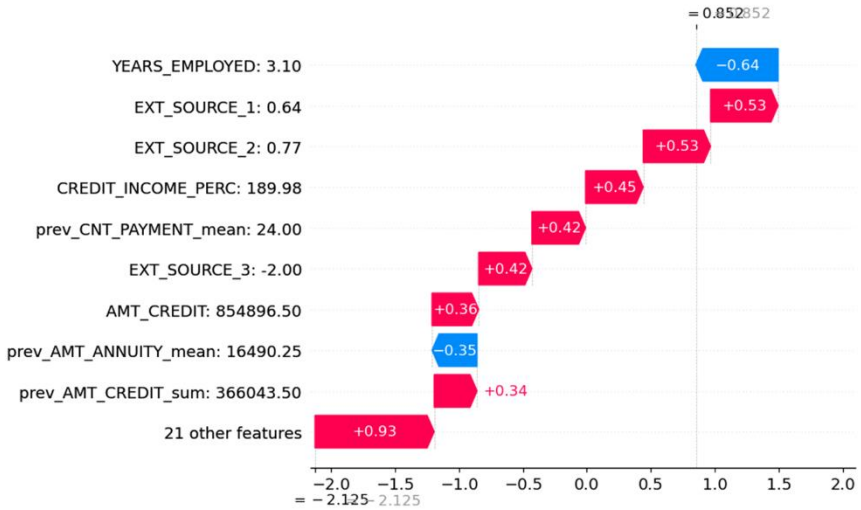
4.1. Les valeurs de SHAP (SHapley Additive exPlanations) globales

Ces dernières permettent de quantifier l'importance relative des variables d'un modèle de machine learning dans ses prédictions globales. Le graphique ci-dessous représente les valeurs de SHAP moyennes pour les 10 premières variables les plus « contributives ».



4.2. Les valeurs de SHAP locales

Contrairement aux valeurs de SHAP globales donnant une vue d'ensemble de l'importance des caractéristiques pour l'ensemble du modèle, les valeurs de SHAP locales se concentrent sur la façon dont chaque variable a contribué à la prédiction particulière pour une observation donnée, soit ici un client. Dans ce projet, elles ont été calculées pour un nouveau client prédit comme étant en difficulté de remboursement (classe 1) et les 9 variables impactant le plus sur la décision sont représentées ci-dessous.



Etant dans le cas d'un client de classe «1 », les valeurs positives (rouge) sont associées à un risque accru d'être classées comme client à risque, alors que celles en bleu ont joué en faveur du client. Ainsi, ce résultat est logique.

NB : La description des 30 variables se trouve dans un fichier pdf nommé « description_30_variables » dans le dossier Datas sur GitHub.

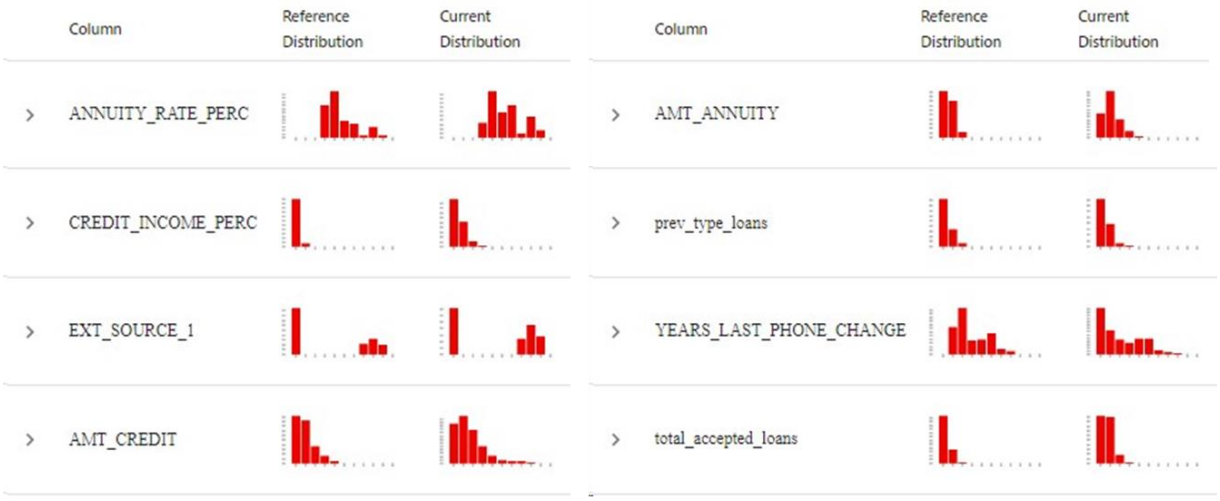
5. Analyse du Data Drift ou Dérive des données

5.1. La stabilité des données, étape préalable au Data Drift

Le DataStabilityTestPreset() est un ensemble prédéfini de tests destinés à l'évaluation de la stabilité des données. Ce dernier inclut des vérifications de la cohérence des données, de la distribution des données ou encore l'absence de valeurs manquantes, afin de s'assurer que les données utilisées en entrée dans un modèle sont stables et conformes aux attentes.
Sur les 93 tests réalisés, 5 ont échoué à cause des 5 variables suivantes : (i) nombre de clients entre les 2 jeux de données (insignifiant), (ii) REGION_POPULATION_RELATIVE présentant une valeur manquante dans le jeu de données des nouveaux clients (insignifiant), (iii) prev_type_loans (mêmes raison et conséquence que pour la variable précédente), (iv) ANNUITY_INCOME_PERC (même constat que précédemment) et (v) AGE où 2 clients sont plus jeunes dans le jeu de données des nouveaux clients, mais avec un écart maximal d'un an (négligeable). Par conséquent, nous pouvons conclure que nos données sont stables.

5.2. Le Data Drift

Le DataDriftPreset() est un ensemble prédéfini de métriques destinées à l'évaluation de la dérive des données. Celle-ci se produit lorsque les caractéristiques statistiques des données changent avec le temps. Ce dernier permet donc de détecter ces changements et d'alerter sur d'éventuels problèmes nécessitant un ré-entrainement des données. Parmi les 30 variables, 8 drifts ont été détectés comme statistiquement significatifs, avec de mon point de vue une dérive déjà non négligeable pour le ANNUITY_RATE_PERC.



Concernant la variable ANNUITY_RATE_PERC, nous pouvons remarquer cette dérive non négligeable entre les moyennes des données ayant servi pour la modélisation (vert) et les nouvelles données (rose).

6. Les limites et les améliorations possibles

6.1. Les limites

Bien que mon ordinateur possède une configuration supérieure à celle demandée pour la réalisation de la formation, ce dernier ne s'est pas avéré suffisamment puissant pour une optimisation optimale du modèle.

De plus, l'analyse de la dérive des données (Data Drift) a montré qu'une dérive était déjà présente pour 8 des 30 variables utilisées, dont une non négligeable.

6.2. Les améliorations possibles

Tout d'abord, l'apport de nouvelles données permettrait de ré-entrainer le modèle afin d'éviter l'accentuation de la dérive des données déjà présentes.

De plus, l'apport d'autres sources de données pourrait être un atout pour la mise au point d'un modèle davantage performant.

Enfin, après avoir réalisé mon optimisation d'hyperparamètres *via* GridSearchCV, je me suis aperçue qu'il existait une autre approche « à priori » plus rapide et plus efficace : Hyperopt. Cette dernière se base sur une méthode d'optimisation bayésienne afin d'explorer de manière intelligente l'espace des hyperparamètres. Ainsi, au lieu d'essayer des combinaisons de paramètres au hasard, Hyperopt utilise les connaissances acquises des essais précédents pour guider la recherche vers les combinaisons qui sont plus susceptibles d'être les meilleures. Malheureusement, faute de temps, je n'ai pas eu l'opportunité de tester Hyperopt mais j'ai au moins pris connaissance qu'il existait une alternative à GridSearchCV que je n'hésiterai pas à expérimenter dans un futur projet.