HOME CREDIT DEFAULT RISK
RÉSULTAT FINAL

 Lien GitHub

Nancy Lelièvre
906 166 672
Samuel Lévesque
111 127 772

Pour le cours GLO-4027/GLO-7027
Analyse et traitement de données massives

Présenté le 17 avril 2019 au professeur

Richard Khoury

Département d'informatique et de génie logiciel
Faculté des sciences et de génie
Université Laval

Faculté des sciences et de génie
Université Laval

Hiver 2019

Table des matières

1 Description de l'algorithme utilisé

Pour ce traitement final des données, nous avions initialement pensé utiliser un modèle à biais plus élevé que lors de nos premiers traitements de données. Toutefois, tel que discuté dans la section 2, les performances préliminaires obtenues avec un modèle SVM étaient beaucoup moins bonnes que celles obtenues avec notre modèle *random forest* présenté dans notre rapport précédent. Pour cette raison, nous avons décidé de tester des modèles de *gradient boosting* qui ont été utilisées par plusieurs utilisateurs Kaggle s'étant démarqué dans la compétition.

Comme dans le cas du modèle *random forest* précédemment utilisé, le module *XGBoost* est basé sur le fait que l'utilisation d'une grande quantité de classificateurs non corrélés permet de faire une normalisation sur les données et de ne pas être trop affecté par les points faibles locaux de chacun des classificateurs individuels. Ainsi, le modèle que nous tentons ici d'optimiser entraîne une grande quantité d'arbres de décision en prenant à chaque fois des échantillons aléatoires de notre jeu d'entraînement obtenant ainsi un ensemble de modèles non corrélés qui font de la classification par vote.

La grande différence introduite par le modèle de *gradient boosting* que nous utilisons ici est la composante de *boosting* lors de l'échantillonnage des données. Cette nouveauté consiste à entraîner chaque arbre de décision sur des données que les arbres de décision déjà existants ont de la difficulté à classifier. En procédant ainsi, les différents classificateurs individuels sont complémentaires et puisque chacun d'eux a un certain pouvoir prédictif, le fait de procéder par vote permet de bien couvrir l'ensemble de l'espace des paramètres tout en évitant d'*overfitter* notre jeu de données d'entraînement. Ce phénomène de régularisation se base sur le théorème du jury de Nicolas de Condorcet.

Les principaux hyperparamètres de ce modèle sont les suivants :

- **Nombre d'arbres** :
  Nombre d'arbres de décision entraînés sur le jeu de données d'entraînement et qui composent la forêt aléatoire. Généralement, plus le nombre d'arbres est élevé, plus on évite les défauts locaux des classificateurs individuels, mais au coût d'un temps de calcul accru.
- **eta** :
  Facteur d'apprentissage pour l'entraînement du classifieur. Ce paramètre est analogue au *learning rate* dans l'entraînement de réseaux de neurones.
- **Nombre minimal d'échantillons par noeud** :
  Définit un seuil qui détermine le nombre minimum d'échantillons dans chaque groupe pour qu'une séparation des données selon un critère soit considérée. Plus ce chiffre est bas, plus l'arbre peut *overfitter* les données, mais plus on pourra couvrir des cas précis avec notre modèle.
- **Profondeur maximale** :
  Profondeur maximale de chaque arbre de décision. Plus les arbres sont profonds, plus les classifieurs individuels sont complexes, mais plus on a de chances d'*overfitter* les données.
- **Objectif** :
  Définit ce qui est ressortit par le modèle. Dans notre cas, on souhaite utiliser l'objectif *binary :logistic* qui fait en sorte que le modèle retourne une probabilité pour une classification binaire, comme c'est le cas dans le contexte de notre projet.
- **Proportion des données à échantillonner par arbre** :
  Permet de spécifier quelle proportion des données d'entraînement est considérée pour entraîner chaque arbre de décision individuel.
- **Métrique d'évaluation** :
  Mesure de performance utilisée pour déterminer quel est le meilleur modèle.

Comme pour le premier traitement de données et tel que discuté dans la section 2, nous utiliserons un algorithme de recherche en grille avec validation à k-plis sur différentes mesures de performance dont l'aire

sous la courbe ROC pour trouver les meilleurs hyperparamètres pour notre modèle. Cette technique consiste à tester une grande combinaison d'hyperparamètres différents et de ressortir les hyperparamètres ayant fourni les meilleures performances selon un critère de performance prédéfini.

Dans notre cas, nous mettons l'emphase sur l'aire sous la courbe ROC puisque c'est la mesure de performance utilisée pour le classement officiel de la compétition Kaggle. Cette mesure met en relation le taux de faux positifs et de faux négatifs selon plusieurs seuils de décision pour notre modèle, trace une courbe comme celles présentées dans la section 2 et calcule l'aire sous la courbe de celle-ci. Cette mesure permet de quantifier la performance du classificateur dans son ensemble en le comparant à un modèle parfait (aire sous la courbe de 1) et un modèle aléatoire (aire sous la courbe de 0.5).

Finalement, dans la section 2.1, on utilise une technique de *resampling* pour rebalancer la proportion des deux classes pour notre modèle appelée *Smote Tomek*. Nous avons choisi cette technique qui combine l'*undersampling* et l'*oversampling* en générant de nouvelles données entre les données de classe connue par partitionnement. De cette façon, on ne fait pas que dupliquer des données existantes ou se débarrasser de données de notre jeu d'entraînement, mais on crée plutôt de nouvelles données synthétiques pour rebalancer le nombre d'observations de nos deux classes. La documentation de l'implémentation de cette technique en Python donne de plus amples détails sur cette technique de rééchantillonnage.

2 Tests et résultats

Pour ce deuxième et dernier traitement des données, nous avons exploré différentes possibilités d'algorithmes. Dans un premier temps, désirant implémenter quelque chose de différent, nous avons utilisé un modèle SVM (*support-vector machine*) linéaire. Malheureusement, ce modèle ne nous a pas permis d'obtenir des résultats particulièrement intéressants. Par la suite, considérant sa grande popularité parmi les participants à la présente compétition *Kaggle*, nous avons opté pour un modèle XGBoost (*extreme gradient boosting*).

2.1 Modèle préliminaire : SVM linéaire

Que ce soit avec ou sans rééchantillonnage, les résultats obtenus avec cette méthode ne sont pas impressionnants. De plus, le temps d'exécution pour obtenir des résultats est énorme en raison de la complexité du modèle ainsi que du nombre d'attributs/observations de nos données.

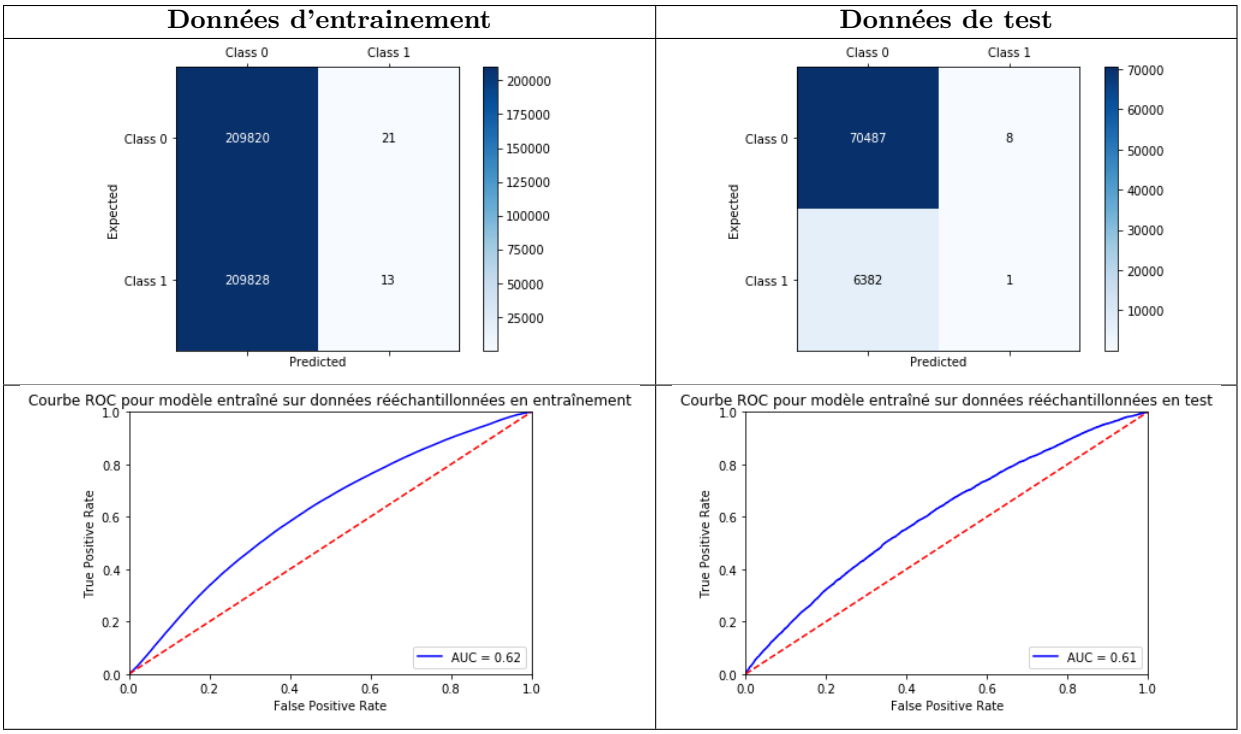
Nous avons utilisé les métriques suivantes pour optimiser notre modèle :

```
scoring_base = ["balanced_accuracy", "recall_weighted", "f1_weighted", "roc_auc"]
refit_base = "roc_auc"
```

Nous avons utilisé les hyperparamètres suivants pour la grille :

```
"penalty": ["l2"],
"loss": ["squared_hinge"],
"C": [0.01, 0.1, 1, 10, 100, 1000]
```

Le modèle optimal est obtenu avec $C = 0.01$ et avec rééchantillonage. Les résultats obtenus en appliquant le modèle entraîné aux données d'entrainement et de test sont les suivants :



Nous obtenons une aire sous la courbe ROC d'à peine 0.61, ce qui ne représente pas une amélioration versus le premier traitement des données par RF (*random forest*). En effet, nous atteignions autour de 0.72 par RF. On remarque aussi que le taux de défaut prédit est pratiquement nul. Dans ce sens, au delà du résultat obtenu pour l'aire sous la courbe ROC et du positionnement dans la compétition Kaggle, cet algorithme ne permet par de répondre au besoin du problème, soit de prédire des défauts éventuels pour des demandeurs de crédit.

Face à ce résultat décevant, nous avons préféré implémenter un algorithme par XGBoost (*extreme gradient boosting*) notamment puisque les participants à la compétition Kaggle avec les meilleurs résultats semblent avoir privilégié cette méthode.

2.2 Modèle principal : XGBoost

Le modèle *XGBoost* est un modèle de classification qui utilise des arbres de décision mais agrémenté d'une méthode de *boosting*. Lors du précédent traitement des données, nous avons été confronté à un problème d'*overfitting*. Ce modèle est sensé limiter ce phénomène. *XGBoost* possède beaucoup d'hyperparamètres, ce qui peut complexifier son ajustement.

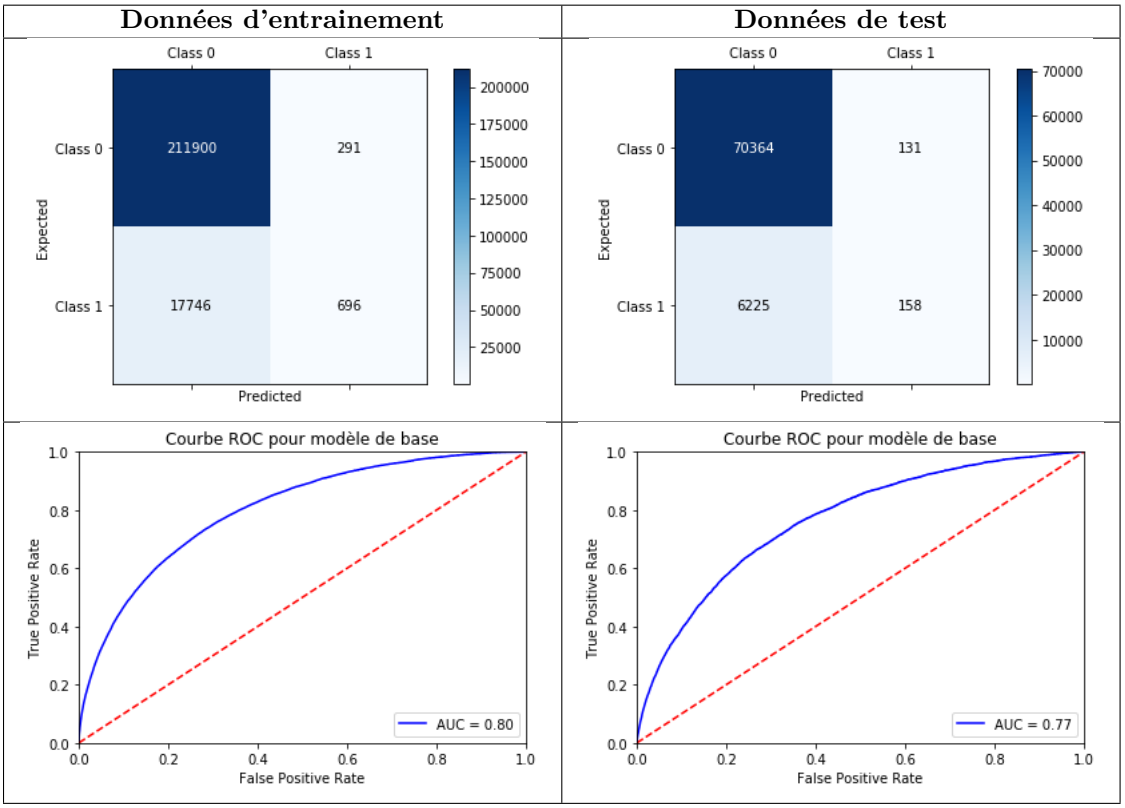
Nous avons utilisé les hyperparamètres suivants pour la grille :

```
param_grid_xgb = {'silent': [1],
                  'n_estimators': [150],
                  'eta': [0.05, 0.075, 0.1],          # learning_rate
                  'min_child_weight': [1],            # default=1
                  'max_depth': [5,7],                 # default=6
                  'objective':['binary:logistic'],
                  'subsample': [0.5, 0.75, 1],        # default=1
                  'eval_metric' : ['auc'],
                  }
```

Le modèle obtenu est le suivant :

```
{'colsample_bytree': 1, 'eta': 0.05, 'eval_metric': 'auc', 'gamma': 0, 'max_depth':
5, 'min_child_weight': 1, 'n_estimators': 150, 'objective': 'binary:logistic',
'silent': 1, 'subsample': 0.75} en entraînement
```

Les résultats des classifications lorsqu'on applique le modèle entraîné aux données d'entraînement et de test sont les suivants :



Nous avons testé plusieurs combinaisons de paramètres pour tenter d'optimiser le résultat du modèle. L'optimisation de ce modèle présente un défi de taille puisqu'il présente plusieurs hyper paramètres et que les combinaisons pouvant être testées explosent facilement. Dans ce sens, les temps de calculs peuvent être très longs. Par contre, nous nous sommes rendus compte que les résultats ne sont pas très sensibles aux modifications sur les hyperparamètres. Le modèle optimal permet d'atteindre une aire sous la courbe ROC d'environ 0.77, seuil qui ne semble pas facile à dépasser en améliorant le modèle. Diminuer la valeur de *n_estimator* engendre une aire sous la courbe ROC moins élevé et il semble que ce soit l'hyperparamètre le plus sensible.

Le modèle ne semble pas présenter de sur-apprentissage. En effet, l'aire sous la courbe ROC pour les données d'entraînement et celle pour les données test sont similaires. De plus, le taux de valeurs positives prédites dans les deux cas est cohérent. Par contre, le taux de défaut prédit, bien qu'il soit semblable pour les données d'entrainement et les données test, n'est pas très élevé. Nous avons possiblement mal calibré le modèle.

En effet, le jeu de données d'entraînement contient environ 8% de défauts alors on aimerait que notre modèle prédise une quantité semblable de défauts, ce qui n'est présentement pas vraiment le cas. Évidemment, on pourrait augmenter la proportion de défauts prédits en abaissant le seuil utilisé pour la classification, mais le faible nombre de défauts prédits avec un seuil de 0.5 démontre que notre modèle n'est pas très polarisant et qu'il est souvent incertain de ses prédictions.

L'aire sous la courbe ROC obtenue est la plus élevée jusqu'à présent. Nous savons que les meilleurs équipes dans la compétition ont obtenu des résultats autour de 0.80, mais ils ont utilisé toutes les données disponibles pour le problème, contrairement à nous qui avons exclu les données supplémentaires pour nous concentrer uniquement sur celles du formulaire d'application.

2.3 Modèle principal : XGBoost pondéré

Bien que les résultats précédents soient supérieurs à ceux obtenus avec la *random forest*, il reste néanmoins que le taux de prédiction pour les défauts demeure petit. Le modèle *XGBoost* permet de considérer le débalancement dans les données en donnant en paramètre le taux de débalancement sous la forme (quantité de 0)/(quantité de 1). Nous avons entraîné à nouveau le modèle en utilisant ce paramètre.

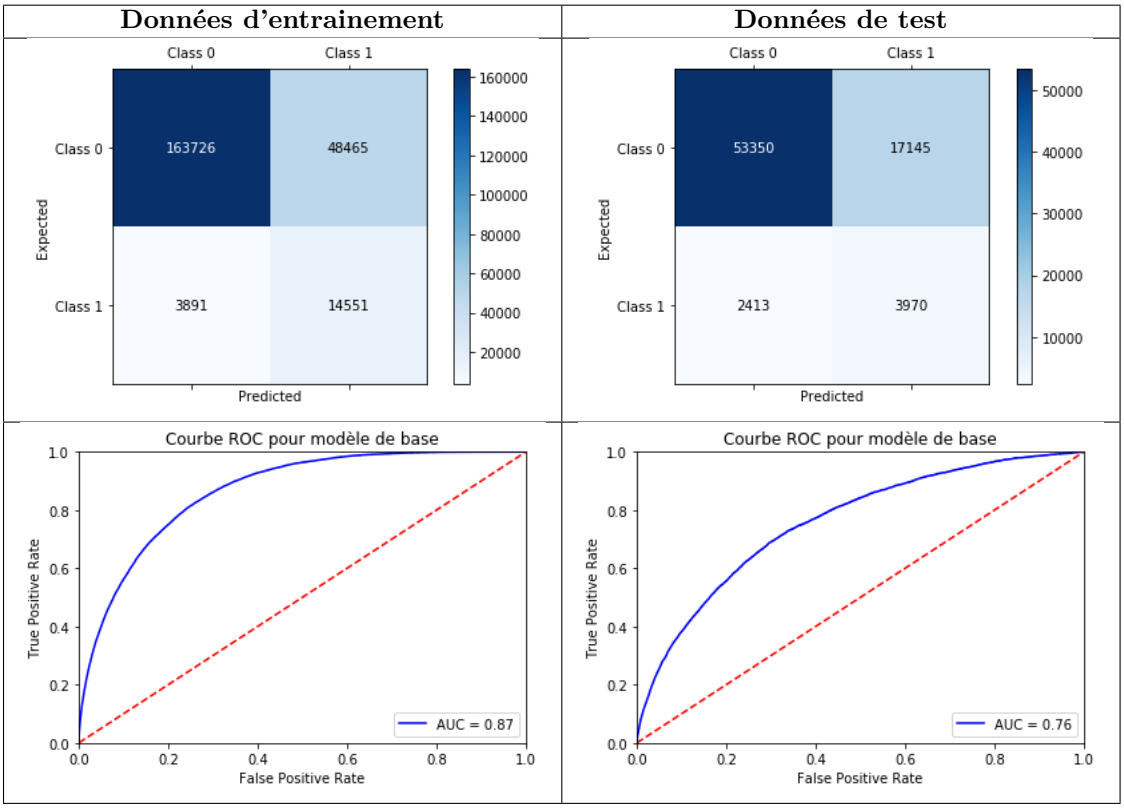
Nous avons utilisé les hyperparamètres suivants pour la grille :

```
param_grid_xgb = {'silent': [1],
                  'n_estimators': [150],
                  'eta': [0.05, 0.075, 0.1],    #learning_rate,
                  'min_child_weight': [1],      # default=1
                  'max_depth': [5,7],           # default=6
                  'objective':['binary:logistic'],
                  'subsample': [0.5, 0.75, 1],      # default=1
                  'eval_metric' : ['auc'],
                  'scale_pos_weight': [scale]
                  }
```

Le modèle obtenu est le suivant :

```
{'eta': 0.05, 'eval_metric': 'auc', 'max_depth': 7, 'min_child_weight':
1, 'n_estimators': 150, 'objective': 'binary:logistic',
'scale_pos_weight': 11.387150050352467, 'silent': 1, 'subsample': 0.75}
```

Les résultats de classification pour les données d'entraînement et de test sont les suivants :



Les résultats sur les données d'entraînement sont encourageants, ils montrent une meilleure capacité de prédire des défauts. Par contre, le modèle devient moins compétent pour prédire des clients qui ne seront pas en défaut, ce qui ne le rend pas nécessairement meilleur versus celui sans pondération. Lorsqu'on s'intéresse aux données test, on remarque le même phénomène. Globalement, nous sommes passé d'un modèle qui ne prédit pas assez de défauts à un modèle qui en prédit trop. Le modèle ne semble pas être en surapprentissage.

L'aire sous la courbe ROC obtenue avec les données de test est inférieure à celle obtenue pour le modèle sans pondération. Considérant qu'il s'agit de la métrique évaluée pour le projet *Kaggle*, nous avons choisi comme modèle final celui sans pondération. Aussi, un modèle qui prédit trop de défauts ne serait pas nécessairement souhaitable dans la vraie vie puisque plusieurs personne se verraient refuser du crédit alors qu'elles ne seraient pas à risque.

3 Attributs importants

3.1 Attributs utilisés

Dans le prétraitement des données, nous avons créé de nouveaux attributs binaires pour transformer nos attributs qualitatifs en attributs numériques. Outre ces transformations, nous avons aussi modifié les valeurs manquantes pour certains attributs lorsque le taux de valeurs manquantes était inférieur à 40 % et la corrélation supérieure à 0.05. Les autres attributs présentant des valeurs manquantes mais qui ne répondaient pas à ces critères ont été supprimés des données.

Dans le prétraitement, nous avions envisagé d'ajouter des variables aux données en les construisant à partir d'autres attributs. Nous pensions pouvoir ajouter facilement le taux d'intérêt des prêts, mais après avoir exploré les données plus en détails, nous nous sommes rendu compte qu'il n'était pas possible de calculer cette variable pour les applications courantes. La variable peut seulement être calculée pour les applications passées, et nous avons décidé de nous concentrer uniquement sur les applications courantes puisque c'est ce que nous tenterons ultimement de prédire avec notre modèle.

Par contre, nous avons construit certains variables qui nous semblaient pertinentes pour prédire les défauts. Ces variables sont les suivantes :

Attribut construit	**Description**
AMT_ANNUITY/AMT_INCOME_TOTAL	Montant du versement versus le revenu de l'emprunteur.
AMT_ANNUITY/AMT_GOODS_PRICE	Montant du versement versus la valeur du bien sous-jacent.
AMT_CREDIT/AMT_GOODS_PRICE	Montant du prêt versus la valeur du bien sous-jacent.
AMT_CREDIT/AMT_INCOME_TOTAL	Montant du prêt versus le revenu de l'emprunteur.
DAYS_EMPLOYED/DAYS_BIRTH	Nombre de jours travaillés versus nombre de jours en vie.
AMT_ANNUITY/AMT_CREDIT	Montant du versement versus montant de l'emprunt.

Les variables construites permettent notamment de connaître le ratio prêt/valeur de l'emprunt, la proportion du revenu qui servira à rembourser le prêt, la proportion de jours travaillés ainsi qu'une approximation du terme de l'emprunt.

Considérant la modélisation choisie, nous utilisons tous les attributs résultant du prétraitement des données plus les attributs supplémentaires construits. On note que tous les résultats présentés dans la section 2 du rapport ont été obtenus avec ces mêmes prétraitements de données.

3.2 Attributs dont la valeur prédictive est importante

Dans le prétraitement des données, nous avions ciblé certains attributs qui nous semblaient plus importants en raison de leur forte corrélation positive ou négative avec la variable cible.

Corrélations positives :

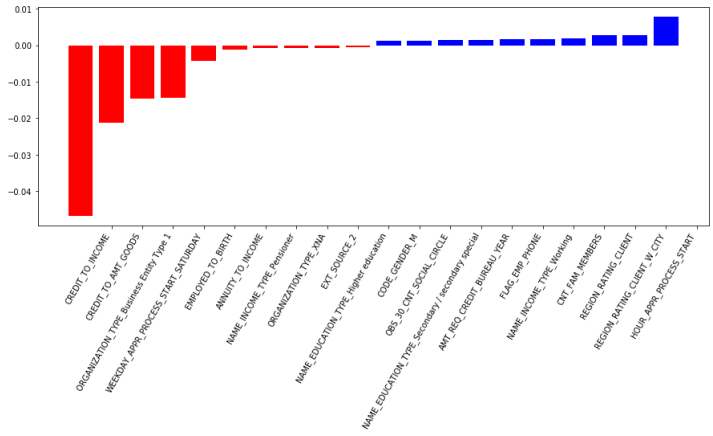
Attribut	Corrélation
DAYS_BIRTH	0.0782
REGION_RATING_CLIENT_W_CITY	0.0609
REGION_RATING_CLIENT	0.0589
DAYS_LAST_PHONE_CHANGE	0.0552
DAYS_ID_PUBLISH	0.0515

Corrélations négatives :

Attribut	Corrélation
EXT_SOURCE_3	- 0.1789
EXT_SOURCE_2	- 0.1605
EXT_SOURCE_1	- 0.1553
DAYS_EMPLOYED	- 0.0449
FLOORSMAX_AVG	- 0.0440

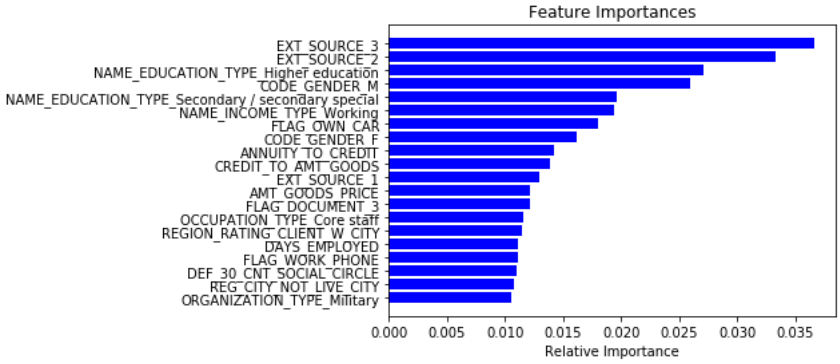
Pour chaque test effectué, nous avons mesuré les attributs les plus importants pour le modèle développé :

- **Modèle préliminaire : SVM linéaire :**



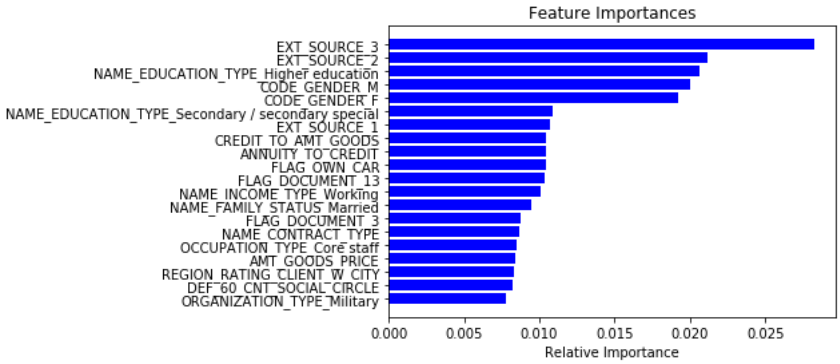
L'attribut le plus important pour ce modèle est clairement CREDIT_TO_INCOME, une variable construite qui représente le montant du prêt par rapport au revenu de l'emprunteur. L'attribut CREDIT_TO_AMT_GOODS, soit le montant du prêt par rapport à la valeur du bien sous-jacent, semble aussi important pour le modèle. Ces résultats semblent logiques puisque ces deux attributs donnent de l'information sur la capacité de payer de l'emprunteur ainsi que le montant de capital initialement investi dans le bien. Les autres attributs ayant une bonne valeur prédictive sont le type d'organisation ainsi que la journée et l'heure de la demande de crédit.

- **Modèle principal : XGBoost :**



L'attribut le plus important pour le modèle est EXT_SOURCE_3, ce qui est cohérent avec le test initial et la corrélation négative calculée. Aussi, EXT_SOURCE_2 est le deuxième attribut le plus important. Ces deux attributs représentent un score normalisé provenant d'une source externe, comme par exemple une cote de crédit. Les attributs ANNUITY_TO_CREDIT et CREDIT_TO_AMT_GOODS que nous avons construits sont dans les plus importants pour le modèle. Le sexe, le niveau d'éducation et l'emploi sont aussi importants.

- **Modèle principal : XGBoost pondéré :**



Les attributs importants sont très similaires à ceux du modèle sans pondération. Les scores normalisés provenant de données externes ainsi que le niveau d'éducation et le sexe sont les attributs les plus importants pour le modèle. On peut remarquer que ces attributs se démarquent versus les autres attributs importants dans le graphique.

En somme, les attributs que nous soupçonnions être davantage explicatifs en raison de leur forte corrélation négative ou positive semblent effectivement importants dans les différents modèles développés. Les attributs construits semblent aussi avoir une valeur prédictive, mais moins importante que les scores normalisés. Le pouvoir explicatif de ces variables confirme qu'il est important d'avoir une bonne connaissance de la problématique pour construire un modèle prédictif performant.

4 Nouveaux problèmes et pistes de solution

Suite à ce deuxième traitement des données, bien que nous ayons augmenté notre résultat pour l'aire sous la courbe ROC, nous sommes encore assez loin des meneurs de la compétition Kaggle. En effet, les gagnants ont obtenu un score de plus de 0.80, ce qui semble plutôt difficile à atteindre selon nos différentes tentatives.

Pour le projet, nous nous sommes uniquement concentrés sur les données provenant des formulaires d'applications pour les demandes de crédit. Par contre, plusieurs autres sources de données, comme par exemple les données sur des prêts passées, étaient disponibles. En utilisant l'ensemble des données, nous pourrions possiblement augmenter notre performance dans la compétition.

Un problème majeur rencontré dans l'élaboration de la solution optimale pour le problème réside dans le nombre élevé d'attributs et d'observations associés au projet. En effet, peu importe le modèle, le temps requis pour l'entraînement est assez élevé. De plus, lorsqu'on désire optimiser les modèles en utilisant *GridSearchCV*, les temps de calculs explosent. Dans ce sens, nos modèles ne sont peut-être pas optimaux, mais nous n'avons pas pu tester une grande quantité de combinaisons d'hyperparamètres.

Pour améliorer notre solution finale, nous aurions aimé raffiner notre optimisation d'hyperparamètres en testant plus de combinaisons et en testant des plus grands domaines pour les hyperparamètres. Aussi, nous aurions aimé utiliser de la recherche aléatoire sur nos hyperparamètres continus de sorte à tester une plus grande variété de valeurs pour chacun des paramètres.

Nous aurions aussi pu tester d'autres familles de modèle telles *KNN* ou des réseaux de neurones pour voir s'ils auraient pu donner de meilleures performances sur nos données.

Finalement, nous avons vu que les attributs que nous avons définis manuellement avaient une bonne valeur prédictive. Nous aurions pu tenter d'en créer de nouvelles ou de se baser sur des recherches dans le domaine du risque de crédit pour ajouter des variables explicatives qui auraient pu aider notre modèles. Nous aurions aussi pu aller chercher des données externes pour compléter notre jeu de données, mais nous avions déjà une base de données assez complète et volumineuse.

5 Comparaison des deux solution proposées

Après analyse des résultats obtenus avec notre algorithme de *gradient boosting*, on remarque des gains de performance importants par rapport aux résultats obtenus avec notre modèle *random forest* développé lors de notre premier traitement des données.

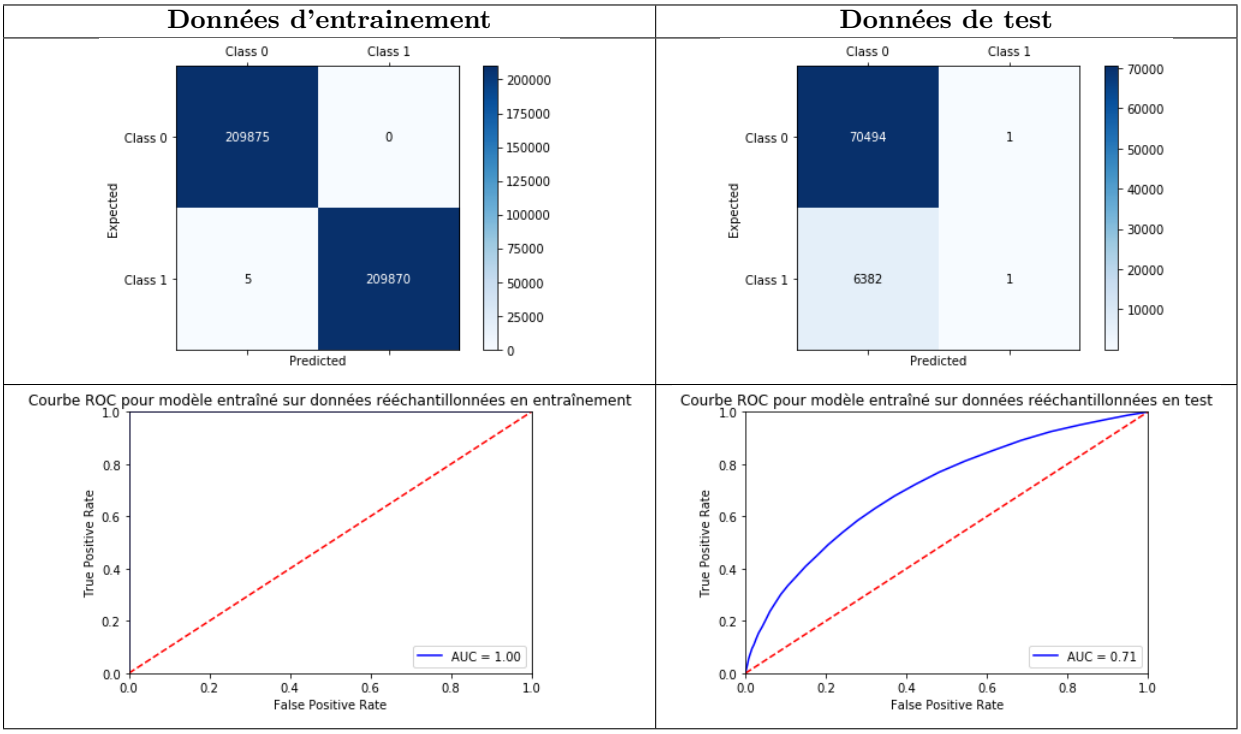
Tout d'abord, si on compare les résultats obtenus avec notre modèle de gradient boosting (voir section 2.2) et ceux obtenus avec notre modèle *random forest* (voir figure 1), on remarque tout d'abord une augmentation importante de l'aire sous la courbe ROC. En effet, avec notre nouveau modèle, cette mesure de performance est passée de 0.71 à 0.77 sur nos données de test. Ce gain de performance est d'autant plus important puisque cette mesure d'évaluation est celle utilisée pour la classification des compétiteurs dans la compétition Kaggle.

Un autre point d'analyse très important se trouve dans la capacité du modèle à *overfitter* les données d'entraînement. En effet, quand on s'intéresse aux matrices de confusion du modèle *random forest* avec les données d'entraînement et de test, on remarque que le modèle arrivait très facilement à apprendre par coeur les données d'entraînement obtenant ainsi un score parfait pour l'aire sous la courbe ROC, mais que ces connaissances se transféraient très mal en généralisation où le modèle ne prédisait pratiquement jamais de défaut.

Avec notre nouveau modèle, on remarque plutôt des performances semblables en entraînement et en test ainsi qu'une plus grande proportion de défauts prédits en test. Cela nous indique que notre nouveau modèle est un peu plus agressif que l'ancien modèle et qu'il permet mieux de repérer les défauts potentiels chez les clients potentiels. De plus, le fait que notre modèle se colle moins parfaitement aux données démontre qu'il capte mieux la distribution réelle des observations dans l'espace des paramètres et qu'on corrige maintenant partiellement le plus grand problème que nous avions avec notre modèle de forêt aléatoire, soit l'*overfitting*.

Ces gains de performance sont attribuables à la composante de *boosting* qu'incorpore le modèle *XGBoost*. En effet, en entraînant chaque arbre de décision de façon à ce qu'il corrige les faiblesses des autres arbres de

FIGURE 1 – Résultats obtenus avec modèle *random forest*



décision, on obtient un modèle qui prédit plus de défauts puisque les premiers arbres de décision sont très conservateurs à cause du débalancement des données. En effet, puisque seulement 8% des données sont des défauts, un arbre ne prédisant jamais de défaut serait correct dans 92% des cas. La mécanique de *boosting* permet ainsi de générer des classificateurs spécialisés dans la classification des 8% de défauts et qui sont plus agressifs que les classificateurs plus généraux.

6 Rétrospective sur le projet

Nous pouvons tirer plusieurs conclusions après réalisation de ce projet de modélisation prédictive.

Tout d'abord, on remarque l'importance de bien comprendre les données avec lesquelles nous travaillons. Effectivement, plusieurs des variables explicatives les plus importantes pour nos modèles étaient des variables que nous avions créées manuellement. Cela illustre l'importance de tenir compte du point de vue *affaires* lors de la création de modèles et non d'appliquer aveuglément des techniques de traitement de données.

Aussi, on peut voir l'importance de faire une bonne recherche pour connaître l'état de l'art du problème auquel on s'attaque et pour faire l'inventaire des pistes de solution à envisager. En effet, les meilleures pistes de solution que nous avons testées proviennent de discussions Kaggle menées par les créateurs des meilleures solutions qui ont été proposées pour la compétition.

Par contre, si nous avions à refaire le projet, nous aurions utilisé la totalité des données pour aller chercher de meilleures performances. En effet, dans l'étape d'analyse des données, nous avions mis de côté une partie des données sous l'hypothèse que leur pouvoir prédictif était trop faible pour la quantité de données supplémentaires que nous aurions dû traiter. On voit toutefois avec le score que nous avons obtenu sur Kaggle que ces données permettent d'aller chercher des performances qui nous auraient permis de monter dans le classement. Effectivement, avec un score (aire sous la courbe ROC) de 0,7556, notre modèle nous permet

de nous placer au 5000e rang de la compétition sur environ 7500 soumissions malgré le fait que nous avons passé beaucoup de temps sur l'optimisation de nos modèles.

Également, après un premier traitement de nos données, nous pensions qu'utiliser un modèle *SVM* à biais plus élevé nous permettrait d'obtenir des performances supérieures à celles obtenues avec le modèle *random forest*, ce qui n'était pas du tout le cas, tel que discuté dans la section 2.1.

De plus, avec la grande quantité de données avec lesquelles nous devions travailler et les nombreux hyperparamètres de nos modèles de *gradient boosting*, nous avons rapidement vu les contraintes computationnelles associées aux projets d'envergure en modélisation prédictive. Cela nous a permis de nous poser des questions sur les avenues de recherche que nous souhaitions tester plutôt que de se disperser et de tester toutes les solutions auxquelles nous pensions.

Pour conclure, ce projet nous a permis de mettre en pratique plusieurs techniques de modélisation prédictive vues en classe. Entre autres, nous avons dû traiter nos données pour contourner les problèmes que le débalancement important des données nous causait et avons dû utiliser un regard critique pour le choix de nos hyperparamètres et de notre modèle final.