# Méthodologie d’entrainement du modèle

## L’enrichissement de la donnée

La base de données de base est constituée de 9 fichiers csv pour un total de 225 colonnes. Il est possible de créer d’avantage d’informations sur les données avec un travail de feature engineering. La première étape est donc de créer de nouvelles variables à partir de celles disponibles.

Nous nous retrouvons avec 389 variables. Les nouvelles variables crées n’ont pas toutes un nombre suffisant de valeurs pour être utilisées. Nous supprimons les colonnes remplies à moins de 80%.

Entrainer un modèle sur autant de variable nécessiterai énormément de ressources. Aussi allons-nous établir l’importance des variables dans l’entrainement des modèles pour ne garder que les plus importante. On parle de « feature selection ».

## Feature selection

La méthodologie est d’estimer l’importance des variables à disposition dans le résultat à l’aide de plusieurs modèles. Nous compterons ensuite pour chaque variables le nombre de modèle qui l’ont estimé importante. Les variables estimées importantes par plusieurs modèles seront conservées.

### Corrélation de Pearson

Cette méthode regarde la corrélation de chaque variable avec la valeur de prédiction.

### CHI 2 avec SelectKBest

La méthode SelectKBest de scikitlearn va sélectionner le nombre demandé des meilleurs variables en se basant sur le score passé en argument.

### RFE avec la régression logistic (wrapper)

La bibliothèque RFE, Recursive FEatures selection, va entrainer un modèle sur un jeu de donnée comportant de moins en moins de variables à mesure que les variables les moins importantes sont éliminé à chaque nouvel entrainement. L’algorithme s’arrête une fois que le nombre de variable demandé est atteint.

### SelectFromModel (embedded): LogisticRegression, Random forest, LightGBM

La méthode SelectFromModel va entrainer un modèle sur notre jeu de données et donner un score d’importance à chacune des variables. On fixe alors une valeur seuil. Si la valeur absolu de la variable est supérieur à ce seuil, la variable est conservée.

### Dernière étape

La dernière étape est de créer un dataframe listant les variables de notre base et comptant la récurrence avec laquelle elles ont été estimées importantes de la part des différents algorithmes. On décide de ne garder que les variables avec une récurrence d’au moins 3. On peut alors filtrer la base en ne conservant que les variables jugées importantes et l’enregistrer en csv.

## Entrainement du modèle

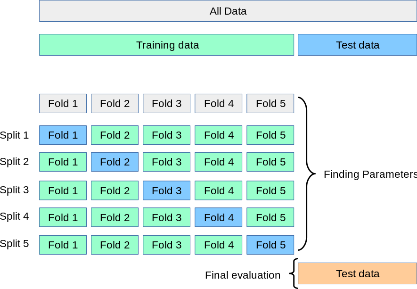
Pour choisir quel modèle utiliser, nous allons comparer différents algorithmes. Le résultat de chacun pouvant varier en fonction de leurs hyperparamètres, il faut dans un premier temps, pour chacun d’entre eux, déterminer leurs meilleurs hyperparamètres. La méthode la plus exhaustive est de procéder par grid search et cross validation.

Grid search :

Pour un algo donné on liste les hyperparamètres que nous voulons personnaliser. Pour chacun de ses hyperparamètres, nous spécifions également l’intervalles de valeurs qu’ils peuvent prendre. La méthode grid search va tester toutes les combinaisons possibles des différentes valeurs de chaque hyper-paramètre. Cette méthode est complète et aucun cas n’est laissé de côté

Cross validation :

La cross validation décrit une méthode d’entrainement d’algorithme dans laquelle le jeu de données d’entrainement est subdiviser en un nombre donné de sous base appelé des « folds ». LE but de cette méthode est de limiter l’overfitting.

L’algorithme va s’entrainer sur l’ensemble des folds sauf un qui servira à calculer le score. Une fois le score calculé, l’algorithme va à nouveau s’entrainer mais cette fois, le folds ayant servi à l’évaluer va faire partie de la base d’entrainement et l’un des précédent folds d’entrainement servira à évaluer la modèle ajusté. L’opération se répète jusqu’à ce que chaque folds ait servi une fois à évaluer l’entrainement du modèle.

Cette méthode est très gourmande en ressource aussi d’autre approches existent.

Méthode bayesian :

Les méthodes bayesians évaluent également différentes valeurs possibles des hyperparamètres mais ne vont pas tester l’ensemble des combinaisons possibles. Elles vont en effet essayer un échantillon de ces combinaisons. Cela a pour effet de diminuer drastiquement le nombre de calcul pour un résultat très correcte.

Optuna

La librairie utilisée s’appelle optuna et applique cette approche bayesian. Son utilisation nécessite de définir une fonction « tune » indiquant le score à utiliser, le modèle à entrainer et le nombre de cycle d’entrainement. Une deuxième fonction « objective » contient les hyperparamètres avec leur fourchette de valeurs, l’instance du modèle à entrainer et le calcule du score.

MLflow

Pour suivre l’évolution de l’entrainement et des valeurs des hyperparamètres, la librairie MLflow a été utilisée. Elle permet de créer des « études » séparée dans lesquelles on peut sauvegarder le modèle entrainé avec les valeurs de ses hyperparamètres et son score.

Une fois tous les modèles entrainés avec la méthode décrite précédemment, nous avons alors un registre des modèles optimisé pour notre problématique. Il est alors facile de les comparer entre eux pour choisir le plus adapté.

Métric utilisée

La métric utilisé dans ce problème a été le roc\_auc qui était plus adapté que « l’accuracy ».

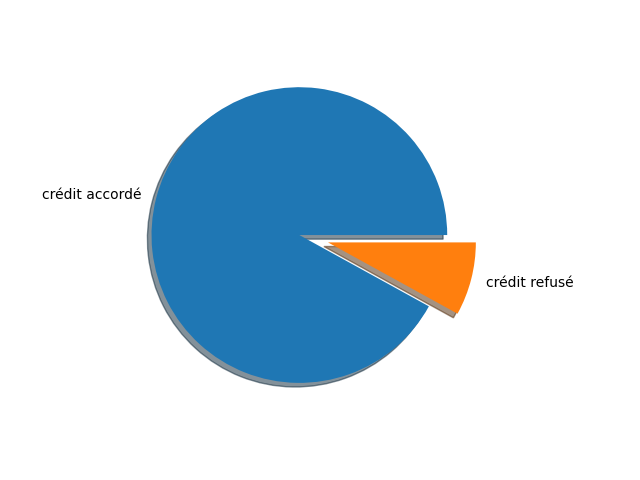
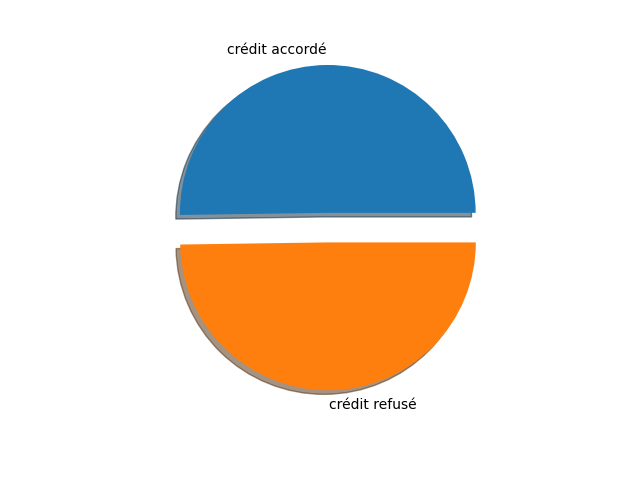
Résultat

Après comparaison, il s’avère que l’algorithme de régression logistique est le plus adapté à notre situation. Ce modèle sera entrainé avec un cout métier, détaillé plus loin, pour comparaison.

# Traitement du déséquilibre des classes

Avant l’entrainement des modèles, une rapide analyse de la variable cible montre une sur-représentation d’un résultat sur l’autre. On parle de déséquilibre des classes. Plusieurs librairies permettent de traiter ce problème.

### SMOTE

SMOTE est une méthode de la librairie ImbalancedLearn qui applique la technique de l’oversampling. Cela signifie que la méthode va apprendre le comportement des lignes de la classe minoritaire et générer de nouvelles lignes aléatoires respectant ce comportement.

Avant (17281/197973) Après (197973/197973)

# Établir une fonction de coût métier

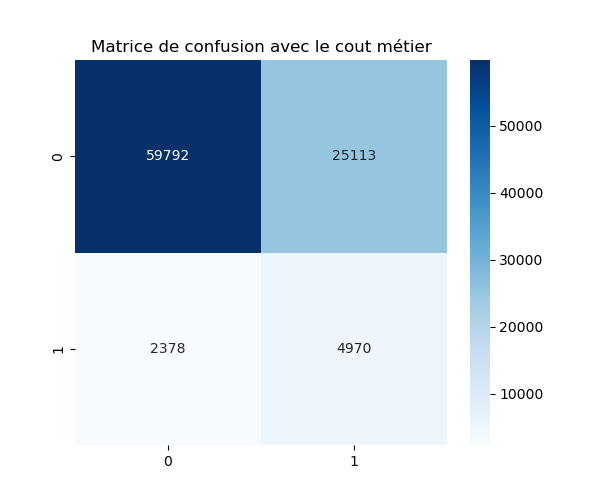
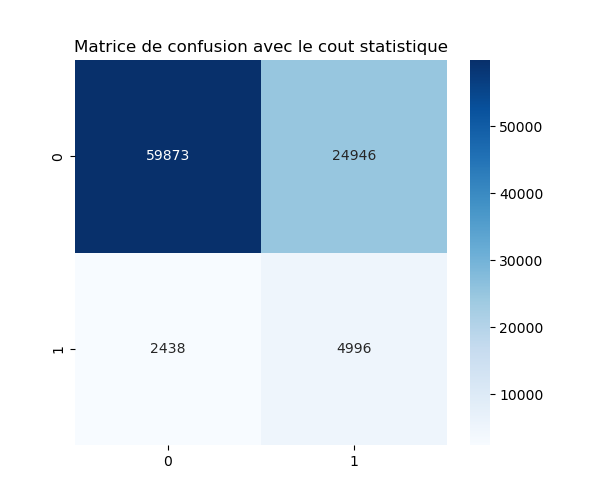
Les modèles ont dans un premier temps été entrainés avec un score statistique. Cependant dans certains cas, il peut être intéressant d’avoir un autre critère d’un point de vue métier. Ici, le modèle est sensé détecter les clients non solvables. En intelligence artificielle chaque modèle génère des vrais/faux positif/négatif. Il est possible de visualiser ces quatre cas simultanément dans une matrice de confusion. Dans notre cas un faux positif décrit un client solvable jugé non solvable et un faux négatif un client à risque non détecté. Chaque fausse estimation coûte de l’argent à la société. Cependant un faux positif génère la perte de l’intérêt sur le prêt alors qu’un faux négatif peut engendrer la perte de tous le prêt en plus des intérêts. Ainsi d’un point de vue métier il est intéressant de donner plus d’importance à ce dernier cas.

Une approche simple est de créer notre propre fonction de coût, qui sera utilisé par notre modèle, dans lequel les faux négatifs ont un ordre de grandeur d’importance supérieur et d’entrainer nos algorithmes pour minimiser ce coût.

Le code de notre nouvelle fonction de coût est le suivant :

Une image contenant texte

Description générée automatiquement On peut voir que le coût consiste à faire la somme des faux positifs et des faux négatifs multipliés par 10. Nous entrainons ensuite l’algorithme exactement de la même manière que précédemment (optimisation des hyper paramètres avec optuna et stockage du modèle dans mlflow) mais en utilisant cette fonction de coût métier comme critère, en demandant à l’algorithme de trouver la plus petite valeur. La pertinence de cette fonction se décide en comparant les deux matrices de confusions :

Nous pouvons voir que notre fonction de coût métier nous permet de diminuer les faux négatifs de 60 ce qui est appréciable. Nous utiliserons donc le modèle entrainé avec la fonction de coût métier.

# Synthèse des résultats

# Interprétabilité locale et globale

# Limites et améliorations envisageables

Mixer oversampling et undersampling. Ne pas forcément arriver à un équilibre des classes parfait. Cout métier plus élaboré que ce qui a été fait.

# Analyse du datadrift