Data Challenge

BEAUCAMP Benjamin
DETREZ Célia

7 avril 2019

Sommaire

1 Présentation du challenge

1.1 L'entreprise

Generali est une entreprise d'assurance créé en 1831 en Italie. Un an plus tard, la société s'est installée en France notamment pour l'assurance des lignes maritimes. Elle diversifie ses services d'assurance et devient l'une des trois plus influentes assurances au monde. Elle lance ainsi en Mai 2017, le projet "The Human Safety Net" afin d'aider les personnes les plus vulnérables. Generali cherche ainsi ces dernières années à améliorer ses techniques d'évaluation de risques afin de proposer des services plus adaptés et de prévenir la mise en jeu d'assurances.

1.2 Objectif du challenge

[1] L'assurance habitation est indispensable lors d'un emménagement. Cette assurance permet d'assurer la protection de la maison en cas de nombreux désagréments comme les inondations, les incendies ou encore les cambriolages. Le challenge propose d'améliorer le processus de facturation du service d'assurance d'habitation, qui se décompose en 2 étapes :

1. Déterminer le nombre de réclamations faite à l'assurance pour une période donnée
2. Prédire le coût moyen d'une réclamation

Nous nous concentrons ici sur le premier point, qui sera simplifié pour le challenge : l'objectif est de prédire si l'assurance habitation sera mise en jeu par l'assuré pendant une période donnée. Il s'agit donc d'un problème de classification à 2 classes : 0 si l'assuré ne met pas en jeu l'assurance durant la période, 1 sinon. Cependant, Generali ne demande pas seulement de prédire la classe mais plutôt la **probabilité** d'appartenir à la classe 1, ce qui est peu habituel dans les problèmes de classification.

La métrique proposée est différente des métriques classiques afin de prendre en compte la prédiction non pas d'une classe mais d'une probabilité d'appartenance à une classe. Il s'agit du coefficient de Gini [1] normalisé, qui est calculé en classant les prédictions par ordre décroissant. Ainsi, seule l'amplitude des probabilités compte et pas leur valeur exacte, ce qui montre que l'assureur souhaite avant tout pouvoir identifier les contrats avec la plus forte probabilité de mise en jeu de l'assurance, parmi tous ses contrats.

1.3 Description des données disponibles

Le challenge fournit un ensemble de données pour l'entraînement avec 26 variables et 10110 échantillons. Le data set de test comporte 3412 échantillons soit 25% de l'ensemble des données totales, ce qui est cohérent. Les variables sont les suivantes :

1. $Identifiant$: numéro client Generali, variable non utilisée pour l'entraînement
2. ft_2_categ : année couverte par l'assurance (de 2012 à 2016)
3. $EXPO$: proportion du temps de couverture pendant l'année
4. $Insee$: code géographique utilisé uniquement pour ajouter des données supplémentaires
5. $target$: cible (1 : mise en jeu de l'assure, 0 : pas de mise en jeu)
6. ft_i_categ : variables anonymisées du bâtiment

Du fait du faible nombre de variables fournies, nous étions fortement encouragés à ajouter de nouvelles données afin d'agrandir le data set, en s'aidant notamment de l'Insee.

1. `https://fr.wikipedia.org/wiki/Coefficient_de_Gini`

2 Méthodologie

Pour travailler sur le challenge en groupe, nous avons utilisés différentes interfaces :

1. Gcloud : utilisation d'une machine virtuelle avec les notebooks pré-installés
2. GitHub : stockage des données, du notebook, et des modèles
3. Google Colab : notebook jupyter en ligne, facilement reliable à une machine virtuelle Gcloud

Nous démarrions alors notre machine virtuelle en effectuant un transfert de port ssh avec notre machine pour par la suite connecter le notebook en local sur la machine virtuelle. Nous avons utilisés des machines avec 4, 8 et 16 CPU. Après chaque entraînement, nous sauvegardions nos modèles en les sérialisant avec `pickle` [9] puis en effectuant un `git push` depuis notre ordinateur.

Nous avons adopté l'approche suivante : ajouter des données dans un premier temps puis tester différents algorithmes sur notre jeu de données.

3 Pré-traitement des données

Afin de préparer les données pour l'algorithme, nous avons utilisé un notebook "Ajout des nouvelles données.ipynb". Sur ce notebook nous effectuons la démarche d'ajout de données par Insee ou département grâce au code INSEE. Les données ajoutées, open sources [4], sont les suivantes :

1. *catnat* : nombre de déclaration de catastrophes naturelles par année et par département
2. *criminalite* : taux de criminalité par année et par département (nombre de crimes/taille de la population)
3. *pprn* : nombre de plan de prévention des risques naturels mis en place par année et par département
4. *pluie* : nombre de jour de pluie avec des précipitations supérieures à $90cm$ [3]
5. $log\_sociaux$: nombre de logements sociaux par département
6. $fuites$: indicateur de l'état des canalisations [2]
7. *entreprise* : nombre d'entreprises comprenant le secteur de la restauration, du commerce, de l'hôtellerie par département
8. *chomage* : taux de chomage dans le département, pour l'année durant laquelle le batiment est couvert [7]
9. rev_med : revenu médian par code Insee, pour l'année durant laquelle le batiment est couvert [8]. Les données de 2016 n'étant pas disponibles [2], nous avons décidé d'estimer la valeur du revenu médian par la moyenne des deux années précédentes.

Ces données nous semblaient importantes car elles reflètent les points que l'assurance couvre ainsi que la situation socio-économique de la zone géograpique dans laquelle se trouve le bâtiment.

Une erreur que nous avons faite est d'avoir supprimé les observations contenant des NaN dans le Train, alors qu'il y en avait aussi dans le Test (qu'on ne peut évidement pas supprimer car il faut effectuer des prédictions pour chaque observation du Test). Les modèles construits sur ce jeu de données n'auraient donc pas été applicables sur notre Test. En effet parfois les NaN sont volontaires et peuvent permettre d'affiner le modèle. On a donc décider de remplacer chaque NaN par la moyenne ou la médiane si la variable était continue et ajouter une nouvelle catégorie "-1" pour les variables catégorielles.

Afin d'appliquer nos algorithmes sur les variables catégorielles, nous avons utilisés deux types d'encoding : LabelEncoder et One-hot encoding. Le premier a l'inconvénient de chercher des liens entre les niveaux des variables (liens qui n'existent pas forcément). Nous avons donc préféré le one-hot encoding, qui crée autant

2. Après avoir pris contact avec l'Insee, nous avons appris qu'elles seront disponibles en Juin 2019

de nouvelles variables que nécessaire pour obtenir des variables binaire. Afin de limiter le nombre de variable, nous avons notamment pour la variable ft_22_categ, correspondant à l'année de construction du bâtiment, regroupé au préalable par groupe de 50 ans. Nous obtenons alors 99 variables.

Afin de faciliter la préparation du Train et du Test et de s'assurer que les variables catégorielles ont toutes les mêmes niveaux dans le Train et le Test, nous avons d'abord ajouté les données sur l'ensemble du jeu de données (Train, Test). Puis nous les séparons à nouveau et supprimons les colonnes *Identifiant* et *Insee* (inutiles dans les modèles).

Nous avons également essayé de mettre en place de la réduction de dimension par ACP sur les variables continues mais les résultats ne sont pas très sensibles du fait de leur faible nombre. La réduction de dimension sur toutes les variables en utilisant la méthode *Factor analysis of mixed data* [10] combinant les méthodes de PCA et l'analyse des composantes multiples (ACM) permet d'obtenir deux axes prépondérants mais l'utilisation dans les algorithmes n'est pas satisfaisante.

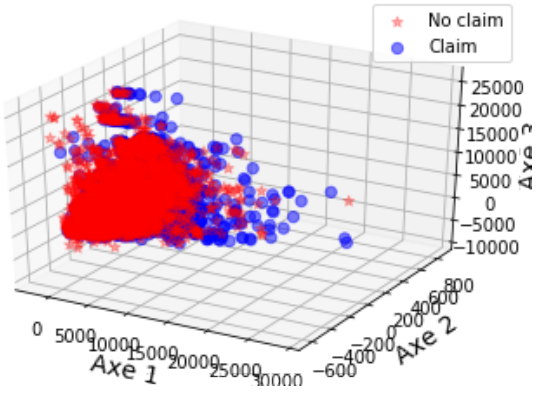

FIGURE 1 – PCA : Représentation sur les trois axes principaux

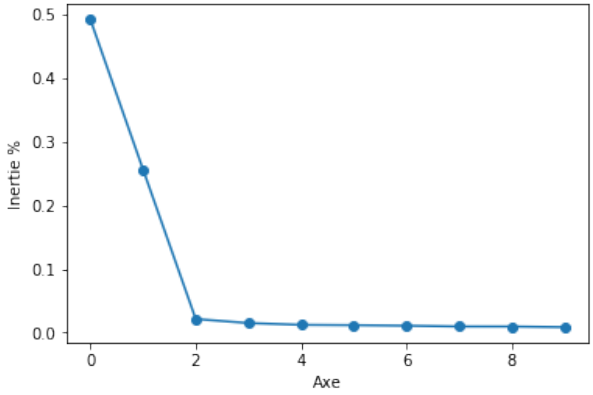

FIGURE 2 – FAMD : Inertie en fonction de l'axe

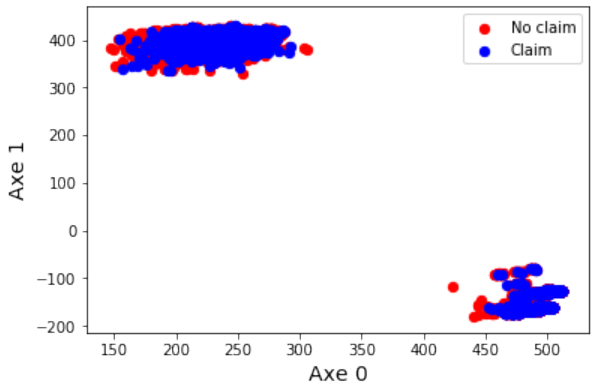

FIGURE 3 – FAMD : représentation en fonction des deux axes principaux

4 Différentes approches

Dans un premier temps, nous avons divisé le jeu de données Train fourni par le challenge en train/test afin d'avoir une idée des performances de nos algorithmes sans avoir à soumettre de prédiction sur le site. Une fois que nous voulions soumettre des prédictions, nous entraînions notre modèle sur tout le jeu de données Train. Les différentes méthodes envisagées sont les suivantes :

1. XGBoost réglé par Gridsearch

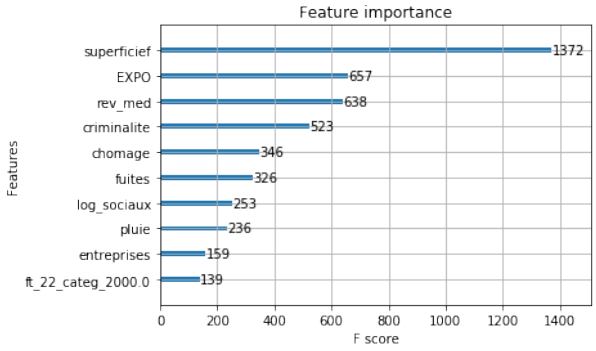

FIGURE 4 – XGBoost : Importance des variables

2. Random forest réglé par Gridsearch

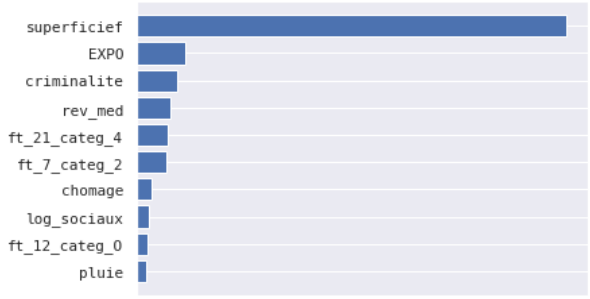

FIGURE 5 – Random Forest : Importance des variables

3. LightGBM [11] réglé par Gridsearch : algorithme de gradient boosting développé par Microsoft, plus rapide que XGBoost

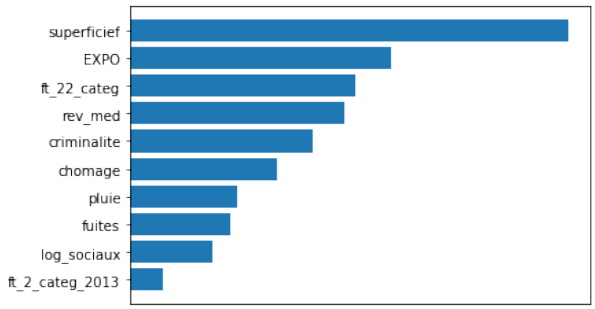

FIGURE 6 – LGBM : Importance des variables

Les GridSearch ont été réalisés par validation croisée 5-folds (pour des questions de performance), en utilisant le coefficient de Gini normalisé comme métrique. Pour ces 3 algorithmes, les scores obtenus sur le test étaient aux alentours de 0.44.

On constate que les variables n'ont pas la même importance suivant l'algorithme utilisé. La variable *superficief* est toujours la plus importante (de loin), mais on voit aussi que les variables que nous avons rajoutées ont un score assez important.

Nous avons également testé des algorithmes alternatifs mais les résultats n'ont pas été satisfaisants :

1. Réseau de neurones : ajout de différentes couches cachées, modification de la profondeur

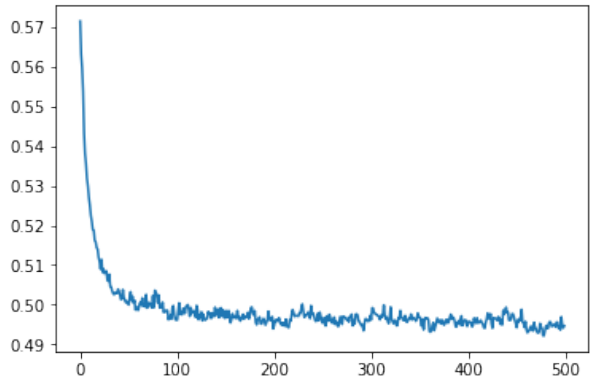

FIGURE 7 – Neural Network : Evolution de l'erreur

   L'algorithme n'arrive pas à diminuer l'erreur (la perte utilisée est `binary_crossentropy`) à partir de 100 époques et ne converge pas de maniére satisfaisante, même en changeant le nombre de couches et le nombre de neurones par couche.

2. Adaboost : essai de Gridsearch pour choisir *base_estimator* mais résultat non satisfaisant

3. SVM : réglé par GridSearch en essayant tous les noyaux fournis par scikit-learn : `linear`, `poly`, `rbf`, `sigmoid`. Les scores étaient bien inférieurs à ceux obtenus par les 3 modèles précédents.

4. Régression logistique : réglé par GridSearch avec une pénalité $\ell 2$. Le score obtenu était un peu plus faible qu'avec les méthodes précédentes, aux alentours de 0.4.

5 Ensembling

Basé sur la lecture de[13], nous avons voulu tester l'ensembling, qui semble être un bon moyen pour améliorer son score sur un data challenge. L'ensembling est une méthode qui consiste à utiliser les prédictions de modèles – dits "de base" – différents afin d'en construire un meilleur. Cette méthode marche d'autant mieux que les modèles utilisés sont très différents, afin que chacun comble les faiblesses des autres.

Nous avons testé 3 de ces méthodes, mais aucune n'a donné de résultat concluant. Cela est probablement dû au fait que nous n'utilisions que peu de modèles et que sur ces 3 modèles, deux étaient très similaires (gradient boosting).

Les définitions des différentes méthodes (notamment le *blending* et le *stacking*) semblent différer d'un article à un autre, nous utilisons donc celles données dans [12].

5.1 Weighted Average Probabilities [5]

C'est la méthode la plus simple : elle consiste à attribuer un poids à chaque modèle, puis à calculer la moyenne pondérée des probabilités d'appartenance à chaque classe.

On commence par séparer notre train en train et validation. On entraine nos modèles de base sur le train et on effectue des prédictions sur le set de validation. Il s'agit ensuite de résoudre un problème d'optimisation pour trouver les meilleurs poids i.e. ceux qui maximisent le coefficient de Gini. Nous avons essayé de les trouver en cherchant sur une grille ainsi qu'en utilisant un algorithme d'optimisation. La première méthode était possible car nous ne possédons pas de beaucoup de modèles et que le nombre d'observations dans le set de validation était relativement faible. Enfin, on entraîne à nouveau les modèles, cette fois sur l'entiereté du train set de départ, puis on effectue les prédictions grâce aux poids trouvés précedemment.

Les résultats obtenus étaient assez décevants, ce qui nous a poussé à tester d'autres méthodes plus avancées.

5.2 Blending [6]

Cette méthode consiste à créer un nouveau jeu de données à partir des prédictions faites par nos modèles. On entraîne ensuite un nouveau modèle sur ce jeu de données.

A partir des jeux de données train, validation et test (sur la gauche de la Figure 8), on commence par entrainer nos modèles de base sur le train. On créé ensuite le nouveau jeu de données train en effectuant des prédictions sur le set de validation, et le nouveau jeu de données test en effectuant des prédictions sur le test. Ces deux nouveaux jeux de données (sur la droite de la Figure 8) nous servent alors à entraîner un dernier modèle.

Nous avons testé la régression logistique et LightGBM pour ce dernier modèle, LightGBM donnant de meilleurs résultats, mais toujours inférieurs à ceux obtenus avec un modèle seul.

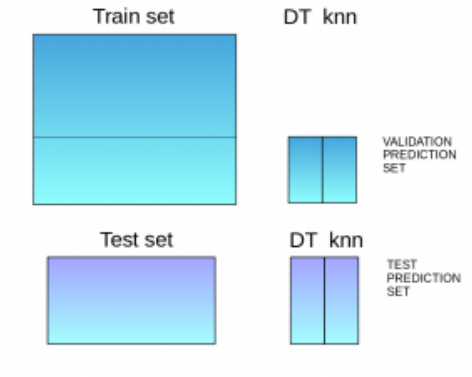

FIGURE 8 – Principe du blending

5.3 Stacking [12]

Le stacking s'appuie sur le même principe que le blending qui est de créer un nouveau jeu de données à partir des prédictions de nos modèles, sur lequel on entraîne un dernier modèle. C'est la manière dont on crée le nouveau jeu de données qui diffère (Figure 9).

Cette fois, on divise le train en (par exemple) 10 parties. On entraîne ensuite nos modèles de base sur 9 des parties du train, puis on prédit sur la dernière. On répète ce processus pour chaque partie du jeu de données train, afin de former notre nouveau jeu de données train.

Pour former le nouveau jeu de données test, on entraîne nos modèles de base sur l'entièreté du train, puis on prédit sur le test. On dispose alors d'un nouveau train et test, grâce auxquels on peut entrainer un nouveau modèle que l'on espère plus performant.

Nous avons ici utilisé un modèle de régression logistique, qui n'a pas non plus fourni de meilleurs résultats qu'un algorithme de gradient boosting seul.

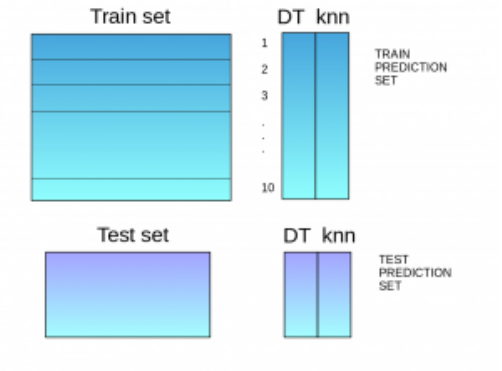

FIGURE 9 – Principe du stacking

$$\text{stack}(x) = \sum_{i,j} a_{i,j} f_i(x) g_j(x)$$

Soit $Identifiant := x$ représentant un assuré dont il faut prédire si il va faire jouer l'assurance $Target := y$, pour les variables définies précédemment f_i et les modèles g_j générés. Les poids $a_{i,j}$ sont variables et permettent de conserver la dépendance des modèles aux données. L'optimisation du modéle se fait donc par régression linéaire.

6 Solution finale et score

Le benchmark du challenge propose un coefficient de Gini normalisé de 0.41 avec XGBoost. Les résultats présentés ici ne sont pas tous obtenus avec le même jeu de données, car nous avons ajouté certaines variables au fur et à mesure que nous avancions et les scores étaient parfois moins bons avec les nouvelles données.

Méthode	Données
XGBoost	Sans fuites et entreprises
LigthGBM	Sans fuites et entreprises
LGBM	Sans entreprises
Stacking de XGBoost, LightGBM et RandomForest	Sans fuites et entreprises
Blending des prédictions de XGBoost, Random Forest et LightGBM	Sans fuites et entreprises
Benchmark	Données initiales

FIGURE 10 – Résultats après soumission sur le site du challenge

La solution finale qui nous classe 2ème actuellement est XGBoost entraîné par Gridsearch sur toutes les données sans celles des fuites et des entreprises.

7 Conclusion

Pour conclure, nous sommes très satisfaits de nos résultats qui nous ont permis d'une part de battre le benchmark, et d'une autre part de nous placer assez haut dans le classement.

Cependant, nous avons été assez déçus de ne pas avoir de résultats concluants grâce à l'ensembling. La difficulté principale de ce challenge était son grand nombre de variables catégorielles, qui nous a bloqué au niveau de la réduction de dimension et nous a empêché d'utiliser des algorithmes classiques de classification comme le k-nearest neighbors, qui semble être souvent utilisé dans les challenges et qui nous aurait certainement permis d'améliorer l'ensembling.

Ce challenge était intéressant car nous avons pu utiliser une métrique non usuelle et ajouter, manipuler et nettoyer nos propres données, ce que nous avons peu fait cette année.

8 Sources

[1] `https://challengedata.ens.fr/challenges/19` *Building Claim Prediction*, Generali, 2019.

[2] `http://www.services.eaufrance.fr` *Services d'eau et d'assainissement.*

[3] `http://pluiesextremes.meteo.fr` *Pluies extrêmes en France métropolitaine.*

[4] `data.gouv.fr`

[5] `https://scikit-learn.org/stable/modules/ensemble.html#weighted-average-probabilities-soft-voting`

[6] `https://mlwave.com/kaggle-ensembling-guide/`

[7] `https://www.insee.fr/fr/statistiques/2012804#titre-bloc-1`

[8] Par année :

2012 : `https://www.insee.fr/fr/statistiques/1895078`

2013 : `https://www.insee.fr/fr/statistiques/2388572`

2014 : `https://www.insee.fr/fr/statistiques/3126432`

2015 : `https://www.insee.fr/fr/statistiques/3560121`

[9] `https://scikit-learn.org/stable/modules/model_persistence.html`

[10] `https://github.com/MaxHalford/prince#factor-analysis-of-mixed-data-famd`

[11] `https://lightgbm.readthedocs.io/en/latest/`

[12] `https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/`

[13] `https://jcohensolal.blogspot.com/2017/02/un-exemple-de-reussite-sur-un-challenge.html`