Rapport TC1 : Projet OTTO Kaggle

Théo Cornille — Cédric des Lauriers — Luc Gibaud

11 novembre 2018

1 Introduction

Le but du projet "Otto Group Product Classification Challenge" est de classifier différents produits selon leurs caractéristiques. La qualité des prédictions sera évaluée selon la log-loss, nous avons donc taché tout au long de ce projet à réduire celle-ci.

2 Données

Trois bases nous sont fournies : la base labellisée d'entraînement « train.csv », la base de test « test.csv » servant à tester nos résultat sur Kaggle, et la base « sampleSubmission.csv » qui nous donne le format de réponse à soumettre.

L'étape d'exploration des données est importante car on ne peut pas entraîner un modèle avec n'importe quelles données, il est donc indispensable de vérifier la nature des variables (le type), s'il y a des valeurs manquantes et la répartition des données. Après cette exploration nous pourrons savoir comment aborder correctement le sujet. Dans notre cas, la base contient 93 features de type 'int64', aucune donnée n'est manquante et la plupart des données sont égales à 0. Il y a 9 classes différentes mais certaines classes sont bien plus représentées que d'autres, par exemple il y a 16122 occurrences de la classe 2 contre 1929 pour la classe 1. Il faut faire attention quand les occurrences des labels ne sont pas réparties uniformément, car le modèle peut avoir tendance à prédire systématiquement la ou les classes sur-représentées. Si c'est le cas il faudra paramétrer les algorithmes en fonction (tuning parameters) , ou encore ré-échantillonner les données. Pour évaluer les performances et optimiser les paramètres de nos classificateurs, nous séparons la base dataframe en 2 jeux de données : un d'entraînement et un de validation. Quand on aura trouvé l'algorithme et les paramètres diminuant au mieux la logloss sur le jeu de validation on réalisera nos prédictions sur Kaggle.

3 Performances des différents modèles

Dans cette partie nous allons passer en revue différents algorithmes, et évaluer leurs performances pour le problème posé. Pour chaque classifieur nous calculerons la logloss, la précision ainsi que la matrice de confusion normalisée. Nous rappelons que la précision correspond à la proportion de valeurs justement prédites, tandis que la logloss tient compte de l'ensemble des probabilités d'appartenance à chaque classe. Chacune de ces métriques sera calculée sur le jeu d'entraînement et sur le jeu de validation.

Il est à noter qu'en apprentissage il est important d'évaluer un modèle sur plusieurs métriques, par exemple un modèle peut avoir une précision très élevée, mais réaliser de très mauvaises prédictions sur les classes sous représentées. La matrice de confusion normalisée nous est donc utile pour évaluer la précision sur toutes les classes.

Les différents modèles ont une logloss oscillant entre 0.48 et 1.7 sur le jeu de validation, l'algorithme qui réalise le meilleur score est l'algorithme Xgboost (eXtreme gradient boosting) , qui très populaire sur Kaggle pour sa grande précision mais qui demande un grande puissance de calcul (cf. Table 1 et Table 2). La grande majorité des compétitions sont d'ailleurs remportées par des modèles utilisant des algorithmes de type boosting.

Les algorithmes dit de boosting ou encore gradient boosting, sont des méthodes ensemblistes qui optimisent une fonction de perte à l'aide du gradient, la fonction de perte est donc différentiable, et contient un paramètre de régularisation afin de contrôler l'overfitting. Le boosting est un modèle additif qui combine plusieurs apprenants faibles, dans notre cas des arbres (Tree Boosting), et minimise la fonction de perte en suivant la direction du gradient lors de l'ajout d'arbre de décision supplémentaires.

TABLE 1 – Evaluation de différents modèles

	logloss_{train}	précision_{train}	logloss_{val}	précision_{val}
Descente de Gradient Stochastique	0.99	0.73	1.0	0.72
Régression logistique	0.66	0.75	0.66	0.75
Forêt aléatoire	0.62	0.82	0.77	0.74
CalibratedClassifieurCV	0.11	0.99	0.48	0.82
Classificateur de vote	0.56	0.80	0.63	0.77
Xgboost	0.10	0.98	0.48	0.82
Xgboost et SMOTE	0.20	0.94	0.55	0.78
Réseau de neurone	0.17	0.93	0.56	0.78

TABLE 2 – Précision des classes avec Xgboost

Classe	1	2	3	4	5	6	7	8	9
Précision	0.52	0.85	0.55	0.54	0.97	0.94	0.68	0.94	0.88

4 Optimisation

L'objectif étant de réaliser la logloss la plus faible sur le challenge nous avons testé plusieurs méthodes susceptibles de la réduire. Nous avons commencé par utiliser les méthodes les plus classiques pour choisir les hyperparamètres, telles que la grid-search et le choix manuel d'hyperparamètres puis nous avons recherché des méthodes plus spécifiques au présent projet.

Ré-échantillonnage : over/under sampling

Comme nous l'avons évoqué précédemment, quand la dispersion des classes n'est pas uniforme les modèles ont tendance à moins bien apprendre les classes sous représentées. L'under sampling consistera à supprimer les valeurs des classes sur-représentées. Le désavantage de cette méthode c'est que l'on perd de l'information, on peut se permettre ce genre de méthode uniquement quand on a beaucoup de données. A l'inverse l'over sampling (SMOTE) consiste à créer de nouvelles valeurs synthétiques à partir des existantes dans les classes sous représentés (ces nouvelles valeurs créer se situent dans le voisinage des anciennes existantes).

Sur notre modèle l'effet de l'over sampling permet une faible amélioration de précision sur les classes sous représentés, en contrepartie d'une faible diminution sur une autre classe, l'effet est donc mitigé.

Feature Engineering

Cette étape consiste à améliorer les features, en construire d'autres, traiter les données manquantes, les normaliser, etc. Elle est essentielle et primordiale pour mettre en œuvre des modèles réalisant des scores élevés. Dans notre cas, puisque les données étaient 'propres', nous avons juste standardisé les features.

Calibration

La calibration de classifieur est une méthode qui permet au lieu de prédire une classe, de prédire la probabilité d'une observation d'appartenir à cette classe possible avec un niveau de confiance élevé. Certains modèles peuvent donner une mauvaise estimation de la probabilité des classes comme c'est le cas pour les random forest classifiers. Ce classifieur tend à prédire beaucoup de probabilités autour de 0.2 et 0.9 et très peu vers 0 et 1 car le bagging établie une moyenne des prédictions sur les différents abres de décision. Les outils CalibratedClassifierCV et calibration_curve de scikit-learn ont donc été utilisé pour calibrer notre classifieur RandomForest et la calibration a donné d'excellents résultats : la log loss est passée de 0.64 à 0.48.

5 Modèle final : un mélange d'autres modèles

On a pu constater en testant différents modèles, que certains classifiaient mieux des classes que d'autres et vice-versa. Pour optimiser nos résultats on peut assembler différents modèles complémentaires en les pondérant. On entraîne chacun des modèles, puis on assemble les prédictions en les pondérant en fonction du modèle. Dans notre cas on combinera le Xgboost sans et avec SMOTE, un CalibratedClassifier et un réseau de neurones à trois couche (avec SMOTE). Pour trouver les poids optimaux on utilise une méthode de descente, on obtient alors comme résultat 0.45 sur le validation test.