Projet TC1 : Challenge Otto Kaggle

Eléonore Bartenlian, Adrien Pavao, Hoël Plantec

1 Introduction

Le challenge Otto Group Product Classification était l'un des défis les plus populaires avec plus de 3,500 participants.
Le groupe Otto est l'une des plus grandes sociétés de commerce électronique au monde, avec des filiales dans plus de 20 pays. Cette société a parrainé ce concours dans le but de trouver un moyen de regrouper précisément leurs produits pour l'analyse des affaires et la prise de décision. Le but était de construire un modèle prédictif capable de faire la différence entre les 9 catégories principales des produits.
Pour ce projet, nous avons décidé de tester une grande variété d'algorithmes et autres méthodes de machine learning.

2 La Métrique

Les soumissions sont évaluées par la fonction loss logarithmique multi-classe :

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij})$$

Avec N la taille de l'ensemble de test, M le nombre de classes, log est le logarithme naturel, y_{ij} vaut 1 si l'observation i est dans la classe j et 0 sinon et p_{ij} est la probabilité prédite que i soit dans la classe j.
Par ailleurs, la somme des probabilités d'un produit donné ne doit pas nécessairement être égale à 1 : en effet, celles-ci sont rééchelonnées en divisant chaque élément d'une ligne par la somme de cette ligne avant que le score ne soit calculé.
De plus, les probabilités prédites sont remplacées par $max(min(p, 1 - 10^{-15}), 10^{-15})$ afin d'éviter les extrêmes de la fonction log.

3 Les Données

Ce dataset est constitué de produits qui sont représentés par 94 features numériques et étiquetés parmi 9 classes.
Il y a 61878 instances dans l'ensemble d'entraînement et 144368 instances dans l'ensemble de test.
Les features et les classes sont simplement labélisés par un numéro : on ne peut donc rien interpréter ni s'appuyer sur des connaissances du monde réel. Les données sont des nombres entiers, généralement proches de zéro.
Par ailleurs, nous avons constaté que les classes étaient assez déséquilibrées : on dénombre 8 fois plus d'instances dans la classe la plus représentée que dans la moins représentée (16000 contre 2000, Fig. 3).
Nous avons également regardé l'importance de chaque feature (selon la Random Forest) et avons constaté qu'il n'y avait pas de feature particulièrement discriminante (Fig 5).

4 Méthodes

4.1 Validation croisée

L'ensemble de validation est créé en échantillonnant aléatoirement 20% de l'ensemble d'entraînement. Nous testons donc les performances de nos algorithmes sur l'ensemble de validation, mais également en faisant des soumissions sur Kaggle. Cela permet d'avoir un score sur un ensemble de test indépendant et d'exécuter l'apprentissage sur l'intégralité de l'ensemble d'entraînement. Ayant constaté que les résultats obtenus par notre validation locale ne différaient que très peu de ceux après la soumission Kaggle, nous avons décidé de ne pas faire de validation plus poussée puisque cela augmente considérablement le temps de calcul.

4.2 Calibration

Lors de nos classifications, nos modèles prédisent l'étiquette de la classe mais également la probabilité associée qui est une mesure du taux de confiance sur cette prédiction. Cependant, tous les classifieurs ne fournissent pas de probabilités bien calibrées, certains étant trop confiants et d'autres pas assez. Étant donné que la métrique du challenge est la log-loss, les prédictions faussement confiantes sont très pénalisées et peuvent altérer le score.
C'est pourquoi nous appliquons des calibrations par régression isotonique aux classifieurs.
La régression isotonique consiste à projeter la fonction non paramétrique dans l'ensemble des fonctions croissantes. La calibration parfaite est la fonction identité : la probabilité prédite colle parfaitement à la fraction observée.

Classifieur	log loss	log loss avec calibration
Decision Tree	9.9073	0.9139
Random Forest	1.4943	0.5870
XGBoost	0.6483	0.6076
Adaboost	2.0262	1.1420
MLP (100,)	0.6576	0.5209
MLP (200,)	0.7910	**0.5089**
LDA	0.9215	0.7603
QDA	5.8836	1.0337
Gaussian naive bayes	7.2519	0.8883
Stochastic gradient descent	2.3360	0.8086
Gradient boosting	0.5996	0.6614
Logistic regression	0.6700	0.6614

Figure 1: Comparaison des classifieurs

On remarque que cela améliore grandement les résultats dans certains cas, par exemple lors d'un bagging où les probabilités produites se rapprochent rarement de 0 et de 1. En revanche, la calibration n'a pas d'effet sur les algorithmes apprenant sur la log loss, comme la régression logistique.

4.3 Réduction de dimension

Nous avons testé une réduction de dimension à l'aide de l'analyse des composantes principales. Nous avons analysé cette méthode selon deux angles : l'évolution de la logloss de la Random Forest en fonction du nombre de composantes de la PCA conservés, et la visualisation de la proportion de la variance de la target conservée en fonction du nombre de composantes de la PCA. Les figures en annexe (Fig. 7 et 8) correspondant à ces méthodes (par la décroissance stricte pour la première, et l'absence de coude marqué sur la seconde) montre qu'il n'est pas nécessaire de faire de la réduction de dimension.

4.4 Over-sampling

En général, les classifieurs sont plus sensibles pour détecter les classes qui sont majoritairement représentées et moins sensibles à celles qui le sont minoritairement. La distribution de nos classes étant assez déséquilibrée, nous avons tester l'over-sampling probabiliste simple. On tire plus probablement les classes les moins représentées de façon à avoir autant de données de chaque classe, avec des duplications. Cependant cela n'a pas amélioré les résultats, cela les a même dégradés dans certains cas.
Il est possible que cela ait provoqué du sur-apprentissage, ou que le déséquilibre des classes n'étaient pas suffisamment marqué pour que l'over-sampling soit significativement utile.
Nous avons également testé le down-sampling, cependant celui-ci a grandement baissé les résultats.
Il serait intéressant de tester un compromis, par exemple échantillonner 8000 exemples par classe.

4.5 Normalisations

Nous avons testé différentes méthodes pour renormaliser nos données : par la moyenne ($x - moyenne(x)$), par la moyenne et déviation standard ($\frac{x-moyenne(x)}{std(x)}$), par la médiane ($x - mediane(x)$), et transformation par $\log(x+1)$.
Cependant, nous n'avons noté aucune amélioration notable.

4.6 Stacking

Le modèle qui nous apporté les meilleurs résultats est une architecture de stacking. Le stacking est une méthode similaire au boosting dans le sens où l'on entraîne et fait fonctionner plusieurs apprenants en parallèle pour fournir leurs prédictions à un "meta-learner". Dans le boosting, l'objectif est de créer de cette manière un apprenant fort à partir d'apprenants faibles et donc d'augmenter drastiquement la capacité du modèle. Dans le stacking, les apprenants étant déjà forts, l'objectif est plutôt de tirer le maximum de profit des spécificités des modèles. Notre modèle comprenait 9 régresseurs sur la première couche:

- Deux méthodes de boosting : gradient boosting et adaboost
- Deux méthodes à base d'arbres: un arbre de décision seul, et une forêt aléatoire
- Un perceptron multi-linéaire
- Une machine à support de vecteurs, avec une descente de gradient stochastique
- Une régression logistique
- Une analyse discriminante linéaire
- Un réseau bayésien naif

Le meta-learner est un algorithme d'XGBoost, choisi car il a l'avantage d'optimiser la logloss dès son apprentissage et ne nécessite donc pas de calibration a posteriori (notons que dans la partie résultats, le stacking 1 est sans calibration, et le stacking 2 comprend une calibration sur XGBoost, la différence des scores n'est pas significative). La diversité de ces méthodes a permis de descendre notre meilleur score de 0.02, nous plaçant ainsi virtuellement à la 1152è place.
Étant donné que la diversité des classifieurs intermédiaires est l'argument d'efficacité principal du stacking, nous avons choisi de ne pas utiliser de calibration sur les modèles du premier étage. En effet cela aurait "lissé" les résultats pour les rapprocher d'une distribution commune, et donc écrasé les disparités que nous recherchions. Il aurait néanmoins été intéressant de tester du stacking en incluant à la fois des modèles recalibrés et non-recalibrés.

4.7 Ajout de features

Pour ajouter au feature engineering utilisé, nous avons créé deux sortes de features :

- Trois features inhérentes à l'échantillon : l'écart-type, la somme, et le nombre de features positives pour l'item considéré
- Des combinaisons de features existantes : somme, différence, produit, rapport, et maximum

Notons que pour des raisons de temps de calcul, ces combinaisons de features ne considéraient que les 10 features les plus intéressantes pour xgboost. Cependant ces paramètres créés n'ont pas amélioré les scores (ils les ont même généralement appauvris).

5 Résultats

Nous avons mis en place des recherches d'hyper-paramètres à l'aide de **grid search**. Voici les résultats obtenus par les différents classifieurs lors de soumissions sur Kaggle :

Modèle	Score Kaggle
Stacking 1	**0.48039**
Stacking 2	**0.48036**
Decision tree	0.90918
Random forest	0.58922
SVM	0.79582
XGBoost	0.60325
Naive bayes	0.89596
MLP	0.52144
Logistic regression	0.66110
LDA	0.75594
Gradient boosting	0.66110
AdaBoost	0.52144
Uncal Gradient boosting	0.60361

Figure 2: Comparaison des scores Kaggle des classifieurs

6 Améliorations possibles

- Rechercher des hyper-paramètres, notamment XGBoost et le stacking (notons que la recherche pour le stacking doit être indépendante de la recherche pour les modèles seuls, puisque rien ne garanti que les optima seront trouvés dans les mêmes régions de l'espace des hyper-paramètres)
- Utiliser la méthode TF-IDF afin d'enrichir le feature engineering
- Effectuer du soft clustering
- Développer des méthodes ensemblistes plus complexes
- Entrainer et comparer différents réseaux de neurones
- Optimiser la construction de features
- Seuiller les probabilités afin d'optimiser la logloss
- Trouver des implémentations des différents classifieurs permettant d'optimiser la logloss dès l'apprentissage

7 Conclusion

Avec le stacking, nous sommes parvenus à obtenir un score de 0.48. Ce résultat est satisfaisant, même si le meilleur score du leaderboard sur Kaggle est de 0.38. Nous avons donc une bonne marge de progression. Une des difficultés du problème était l'opacité des données : nous ne pouvions pas utiliser notre expertise dans un domaine pour savoir quelle direction prendre dans le but de le résoudre. Toutes les informations s'obtenaient donc avec des études statistiques et par l'exploration des diverses possibilités ; nous étions tels des aventuriers dans cette jungle de données.

Annexes

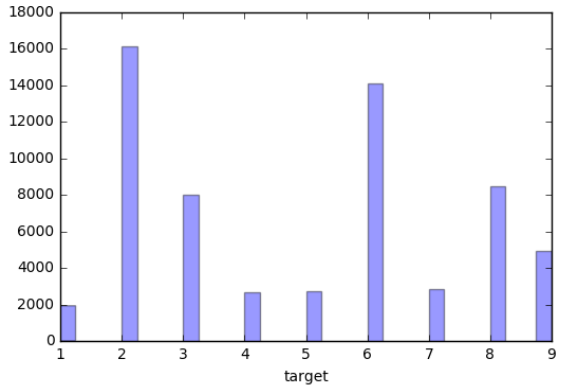

Figure 3: Distributions des classes

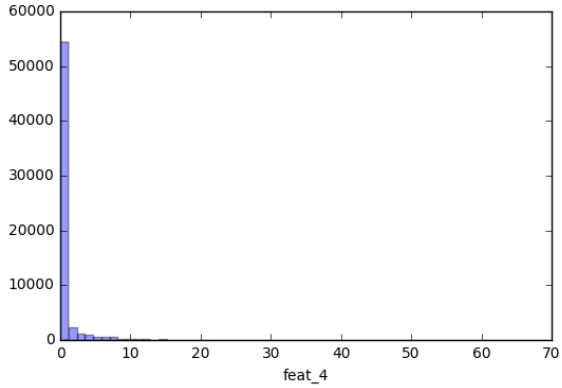

Figure 4: Exemple de distribution d'une feature : features_4

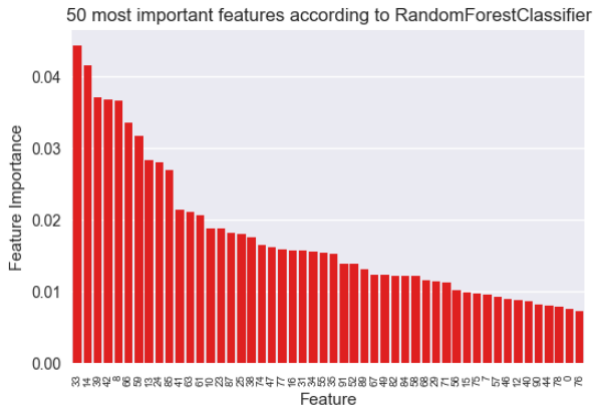

Figure 5: 50 features les plus importantes selon le Random Forest

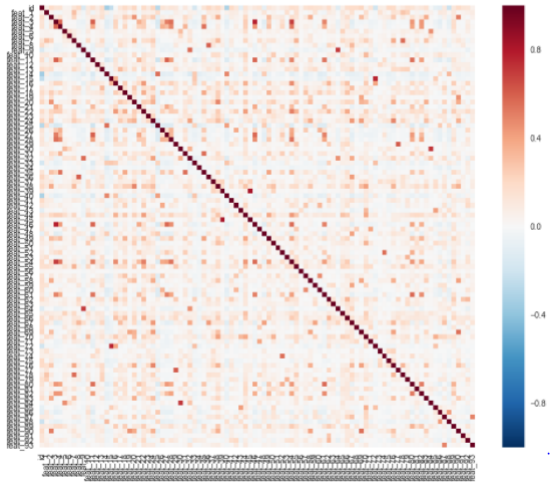

Figure 6: Matrice de corrélation des features

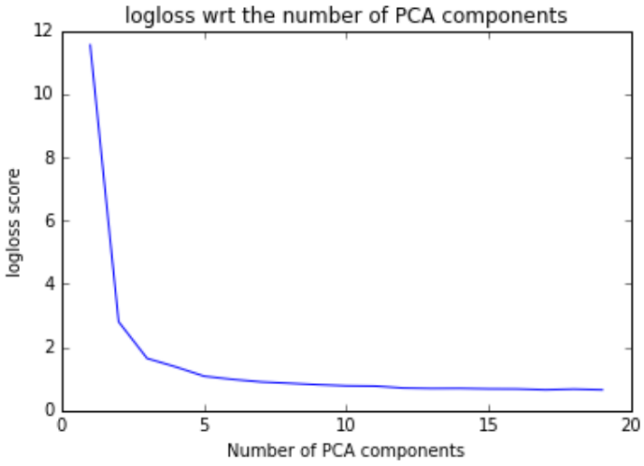

Figure 7: Log loss d'un arbre de décision en fonction du nombre de composantes principales prises en compte

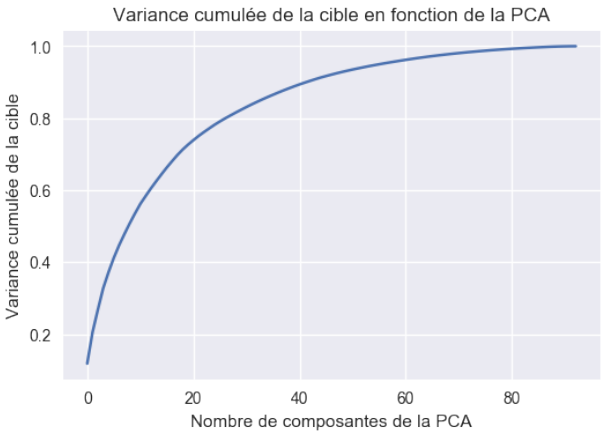

Figure 8: Graphique de la variance cumulée de Y en fonction du nombre de composantes de la PCA

References

[1] T.Chen, C.Guestrin
Xgboost: a scalable tree boosting system

[2] S.B.Kotsiantis
Supervised machine learning: a review of classification techniques

[3] S Lichtenstein, B Fischhoff, LD Phillips
Calibration of probabilities: the state of the art