UMONS

Faculté des Sciences

Big Data Analytics I

Rapport de projet

Sam Boosko
Rémy Decocq
Dimitri Waelkens

Année Académique 2018-2019
Master en Sciences Informatiques
Faculté des Sciences, Université de Mons

1 Introduction

La jeu de données fourni a été construit lors de campagnes marketing menées par un organisme bancaire, sous la forme d'appels téléphoniques vers de potentiels clients. Pour chaque personne sondée, il est renseigné si oui ou non, à la suite de cet appel, elle a souscrit à un dépôt bancaire à long terme dans ladite banque. Le but de la compétition est de prédire si ce sera le cas pour de nouveau client en se basant sur des variables mesurées identiques. Le jeu d'entrainement contient 30436 observations (dont le résultat "a souscrit" est connu et est repris par la variable y), tandis ce que le jeu de test (dont le y nous est sciemment pas communiqué) est séparé en deux et contient au total $10182 \times 2 = 20364$ mesures.

Les données sont des mesures de variables de deux types : celles relatives au client lui-même (données personnelles) et celles relatives aux potentiels sondages sur le client durant les campagnes. On a donc :

Variables explicatives

`Persos` :

- *age*, (type de) *job*, statut civil *marital*, niveau du milieu d'éducation *edu*
- *default* a une défaillance de crédit, *housing* sous prêt immobilier, *loan* a un prêt personnel

`Campagnes` :

- *contact*, *month*, *day_of_week* : type de communication, mois et jour de la semaine du dernier contact
- *campaign* : nombre de contacts établis durant la campagne correspondant au jeu de données
- *pdays* : nombre de jours passés depuis le dernier contact d'une campagne précédente
- *previous* : nombre de contacts déjà établis avant cette campagne
- *poutcome* : résultat pour ce client suite à la campagne précédente

Variable expliquée

- *y* : le client a ouvert un dépôt à long terme dans l'organisme bancaire

2 Méthodologie

2.1 Analyse des variables et observations

Une variable que l'on peut remettre en question serait *default*. Effectivement, si on regarde la proportion des valeurs pour cette variable catégorique, on a 75.24% de 'no', 24.74% de 'unknown' et moins de 0.01% de 'yes' (3 observations). Une variable qui est presque binaire de la sorte avec une valeur à la sémantique inconnue apporte de la confusion et ne permet pas d'expliciter rationnellement une information. Elle sera donc omise.

Un autre problème peut être mis en avant pour la variable *pdays*. Bien qu'elle soit numérique, une valeur de 999 renseigne 'le client n'a jamais été contacté dans une campagne précédente' (et donc le nombre de jours passés depuis le contact lors de la dernière campagne est indéterminé). On utilise donc une valeur numérique pour signifier quelque chose qui n'est pas quantifiable (au contraire des valeurs pour les clients déjà contactés qui s'étendent de 0 à 11 jours). Un compromis pour avoir une meilleure sémantique serait une variable catégorique, considérant 'recent' si le nombre de jours est < 6, 'late' s'il est ≥ 6 et 'never' pour les valeurs 999. Notons que cela se fait au prix d'une perte d'information, mais le nombre conséquent d'observations pour lesquelles *pdays* vaut 999 (30329) nous y motive.

On constate également des incohérences en considérant la variable $previous$ avec $pdays$. Théoriquement, si $pdays = 999$, alors le client n'a jamais été contacté donc $previous$ (indiquant le nombre de contacts ultérieurs toutes campagnes confondues) doit valoir 0. Or, 1316 observations ne satisfont pas ce prédicat (soit 4% du jeu de données) parmi lesquelles 108 ont un réponse positive pour la variable expliquée y. Ces observations sont importantes, effectivement seulement 2342 parmi les 28094 observations affichent une réponse positive pour y. Pour que notre modèle soit efficace, il faut tenir pouvoir tenir compte de toutes ces précieuses observations.

2.2 Sélection du type de modèle

Comme les prédictions à faire sont sur la variable *y* qui est binaire, on s'intéresse aux méthodes de classification. Trois approches ont été considérées : LDA, la Régression Logistique et les Arbres de décision. La première semble assez peu adaptée : nous ne pouvons pas affirmer rentrer dans ses hypothèses de distribution gaussienne sur les prédicteurs et de séparation claire des classes, qui ne seraient qu'au nombre de 2 (ce qui n'exploite pas la force de LDA). La seconde a été sélectionnée dans un premier temps, dans l'optique d'en améliorer les résultats avec les méthodes de Bagging en implémentant les arbres par la suite si le temps le permettais. Effectivement, la **Régression Logistique** semble tout indiquée pour notre problème : variable expliquée binaire, correspondance avec la fonction d'erreur *LogLoss* utilisée pour évaluer nos résultats, dummification des variables catégoriques aisée.

2.3 Sélection des prédicteurs pertinents

Une fois les premiers modèles construits, il est apparu évident que certaines variables n'y étaient vraiment pas significatives. Afin de sélectionner un ensemble optimal de variables apportant de l'information au modèle parmi les 14 disponibles, les technique de **Stepwise Selection** a été employée. En l'opérant dans les deux sens, cela permet de déduire un ensemble fort de variables importantes sur lesquelles reconstruire le modèle de régression logistique.

2.4 Séparation du jeu de données

Comme énoncé à la section 2.1, seulement 2342 observations présentent une réponse positive, ce qui ne constitue que 7% du jeu total. C'est problématique, car notre modèle risque de ne pas les prendre suffisamment en compte car noyés dans la masse (ie. le modèle overfit sur des observations majoritairement telles que $y = 0$). Construire le modèle sur un jeu plus balancé pourrait offrir de meilleurs résultats. Une méthode a donc été mise en place pour séparer le jeu de données en *training/validation* en tenant compte de cela, dont la procédure est :

`SepDataset`, paramètres numériques *prop_ok* et *balance*

1. Diviser le set en *obs_no* et *obs_ok* respectivement les observations telles que y=0 et y=1
2. Constituer le **validation set** : prendre $\frac{size(obsok)}{prop}$ observations de *obs_ok* et un même nombre dans *obs_no*
3. Considérer *obs_remaining* l'ensemble des observations n'étant pas dans le validation set, prendre parmi elles toutes les observations y=1 qu'on désigne par *obs_remaining_ok*
4. Constituer le **training set** : l'union de *obs_remaining_ok* et *size(obs_remaining_ok)* $\times$ *balance* observations dans *obs_remaining* telles que y=0

On obtient alors un validation set de taille limitée contenant $\frac{size(obsok)}{prop} \times 2$ observations, dont la moitié sont telles que y=1 et l'autre y=0. Le training set contient d'une part toutes les observations restantes du set telles que y=1, et d'autre part un nombre d'observations telles que y=0 égal à *balance* $\times$ la taille de cet ensemble. Donc plus *balance* est faible, plus il y aura un nombre équivalent de y=0 et y=1 dans le training set. Le training et validation set sont bien distincts.

2.5 Validation du modèle

Afin de vérifier la cohérence de notre modèle et estimer l'erreur de test, ainsi que d'autres statistiques en découlant, la Cross-Validation est la méthode qui est employée. La procédure se présente de cette façon :

`CrossValidate`, paramètres *nbrCV* et *nbrFolds*

1. itérer de 1 à *nbrCV* en générant *nbrFolds* partitions du dataset aléatoirement à chaque fois
2. Dans chaque itération, itérer sur chacun des *nbrFolds* en considérant un comme l'ensemble de test, et appliquant `SepDataSet` sur l'ensemble formé par les *nbrFolds-1* autres.
3. Lancer le modèle avec les prédicteurs fixés sur l'ensemble de training retourné par `sepDataSet`, évaluer l'erreur sur l'ensemble de test
4. Calculer une moyenne des erreurs sur les *nbrFolds* itérations, à partir des *nbrCV* moyennes ainsi calculées, afficher leur distribution

3 Résultats et discussion

4 Conclusion