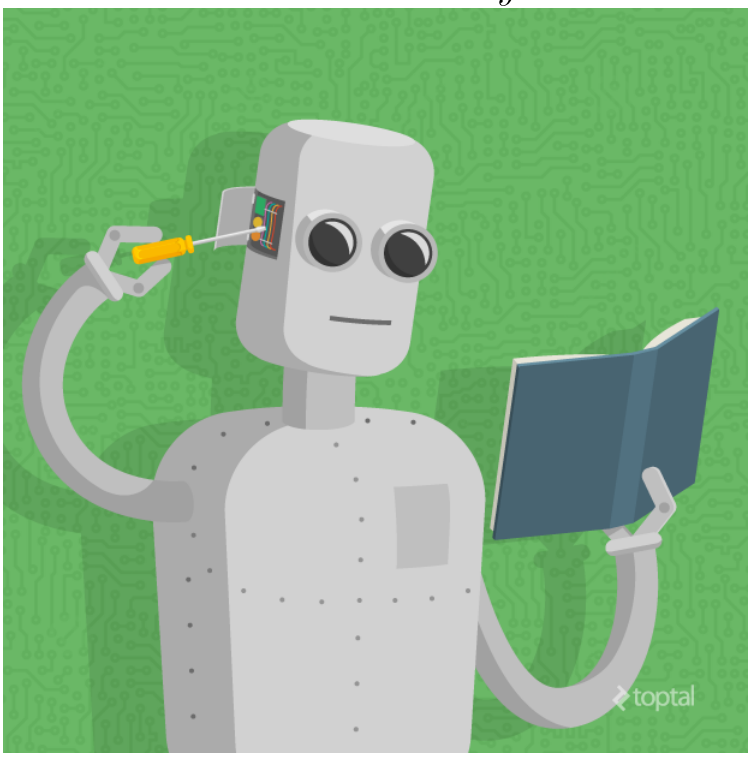
ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

Projet d'apprentissage

Prédire le comportement client pour une garantie accident de la vie.

Abdellah Sabry



avec
Monsieur Denis Oblin

7 janvier 2018

Table des matières

Exploration des données

L'exploration des données est souvent très importante dans un projet de Machine Leanring, car elle permet de voir les corrélations évidentes entre les variables et leurs importances avant même de creuser le problème. J'ai alors travaillé sur deux étapes : la DataViz et l'apprentissage non-supervisée.

1.1 DataViz

Pour la DataViz, j'ai essayé de voir les corrélations entre les variables, ceux qui sont le plus corrélés avec la variable *target*, etc. L'information la plus importante trouvée est que la variable *target* est non équilibrée : seulement 3.52% sont positives (sont des 1). Je n'ai pas trouvé d'autres informations très importantes sauf une légère corrélation entre la variable *age_prospect* et la variable *id*. Plusieurs visualisations de données *DataViz* sont jointes avec ce rapport.

1.2 Non-supervisé

Ensuite, j'ai survolé les méthodes non supervisées qui pourraient donner des informations intéressantes sur la base. J'ai commencé par un K-means, j'ai donc commencé à chercher le nombre optimal de cluster tout en le validant par la silhouette. J'ai appris durant cette étape que l'*id* est une variable très importante, car en gardant cette variable, les clusters sont bien formés (avec une grande valeur d'inter-classe), alors qu'en la supprimant, les observations

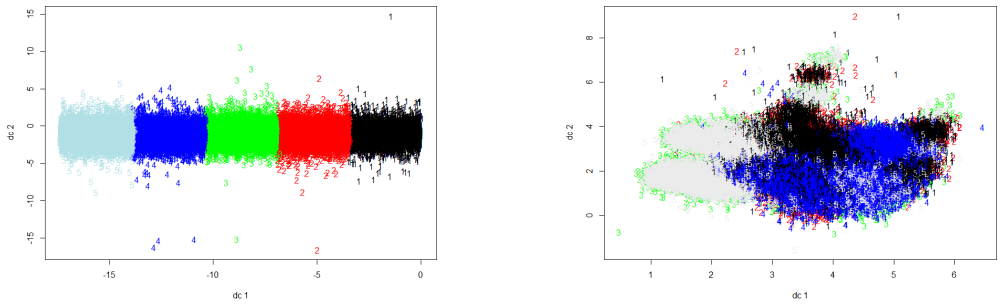


FIGURE 1.1: K-means avec 5 clusters, avec ID vs Sans ID

	PC64	PC65	PC66	PC67	PC68	PC69	PC70
Standard deviation	0.98720	0.98017	0.97479	0.97191	0.9608	0.95366	0.95038
Proportion of Variance	0.00792	0.00781	0.00773	0.00768	0.0075	0.00739	0.00734
Cumulative Proportion	0.82936	0.83717	0.84489	0.85257	0.8601	0.86747	0.87482
	PC71	PC72	PC73	PC74	PC75	PC76	PC77
Standard deviation	0.94641	0.93957	0.92681	0.91947	0.9078	0.90357	0.89280
Proportion of Variance	0.00728	0.00718	0.00698	0.00687	0.0067	0.00664	0.00648
Cumulative Proportion	0.88210	0.88927	0.89626	0.90313	0.9098	0.91647	0.92295

FIGURE 1.2: Variables expliquées par les vecteurs PCA

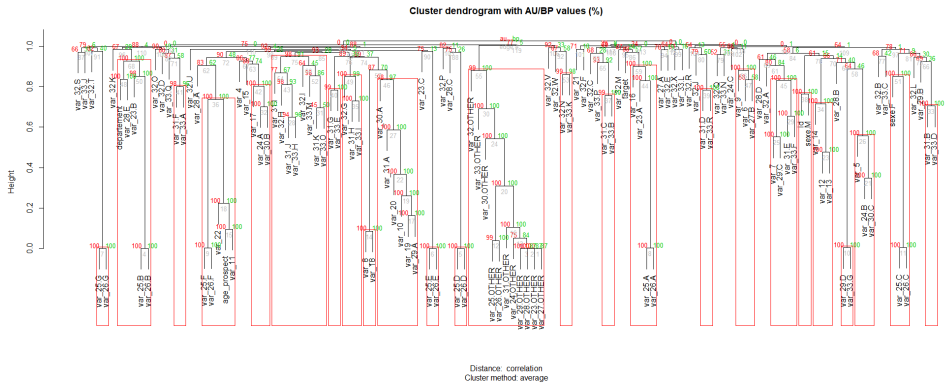


FIGURE 1.3: corrélation hiérarchique entre les variables

se mélangent un peu partout comme nous pouvons le voir ici 1.1. Plus de graphiques seront disponibles avec ce rapport, notamment les différents plots de silhouettes correspondant au nombre de clusters différents. En créant des partitions dans les données, j'aurais espéré trouver une (voire plusieurs) partition ayant des targets positives seulement ou négatives seulement, mais malheureusement tous les clusters ont une distribution équilibrée du target (par rapport à la base de données, c'est-à-dire environ 3% de 1) . J'ai enchaîné avec un PCA après pour voir si je pourrais réduire la dimensions de la base de données avant de commencer l'apprentissage, mais les variances étaient -étonnamment- bien distribué sur toutes les variables. Pour expliquer 80% des variations dans les variables, le PCA a utilisé 60% des variables comme nous pouvons le voir ici 1.2, ce n'est pas une bonne idée de passer par cette méthode avant l'apprentissage supervisé.

Finalement, j'ai utilisé une méthode de cluster hiérarchique pour créer des groupes de **variables** (et non pas des observations) proposée par Ryota Suzuki et Hidetoshi Shimodaira. Cette méthode permet de voir les corrélation entre les variables avec les p-values (AU). Par exemple, les variables $var\_25.B, var\_26.B, var\_23.B, var\_28.E$ et *departement* sont importants ensemble. Alors qu'en leur ajoutant $var\_32.K$, leurs importances diminuent brusquement (de 100 à 68) comme nous pouvons le voir sur la Figure 1.3.

Modèles utilisés

Avant d'attaquer les modèles utilisés, j'expliquerai ma démarche pour la préparation des données. J'ai commencé par joindre les deux bases *train* et *valid* pour appliquer les transformations sur toutes les données. Puis j'ai manipulé les NAs de la base : i) les NAs des colonnes quantitatives sont remplacés par la médiane de la colonne, ii) les NAs des colonnes qualitatives sont remplacés par la chaine de caractères "$Ex.Na$". Puis j'ai cloné cette base (contenant valid+train, sans NAs), pour créer une nouvelle base contenant des $one - hot\ vectors$ à la place des colonnes qualitatives. Cette base sera utile pour des modèles n'acceptant que les entrées numériques comme le glm et le Xgboost. Finalement, j'ai re-divisé les deux bases (original et dummies) en train et valid, pour ensuite diviser aléatoirement les deux train en gardant 75% pour l'entrainement du modèle et 25% pour son test.

2.1 Logistic

La régression logistique est le modèle que j'ai l'habitude de lancer en premier. Même s'il s'avère être -très- simple et donc parfois non performant, il donne une approximation de la prédiction à laquelle on pourrait s'attendre pour cette base de données. J'ai donc eu pour mon premier modèle (nommé glm1) contenant toutes les variables une AUC de **64,59%**. J'ai lancé un autre modèle (nommé glm2) en gardant seulement les variables ayant un p-value très faible (p-value ≤ 0.05), et donc en gardant les variables les plus importantes (que le premier modèle a jugé ne pouvant s'en passer avec un seuil de risque de 5%) mais l'AUC est tombé à **60,55%**. Ce résultat n'est pas choquant, car j'ai choisi de garder des variables "importantes", mais celles-ci changent d'importance à chaque combinaison. Néanmoins, je préfère souvent commencer par ce modèle avant d'attaquer les modèles de *features selections*. Il en existe trois : i)Backward : commencer avec toutes les variables et retirer la pire, ii)Forward : commencer avec 0 variable dans le modèle, et ajouter la meilleure et iii)Stepwise la fusion des deux : avancer trois pas en avant (forward), et reculer un pas en arrière (backward) par exemple. J'ai alors utilisé le stepwise (modèle nommé glm.stepwise) sur un échantillon aléatoire des données de 20 000 observations. Je remonte ma courbe ROC avec **63,58%** d'AUC, qui est moins bon que le premier modèle mais j'ai maintenant une idée des variables importantes dans la base de données. À l'étape suivante, j'utilise une extension du *glm* très efficace nommé *elasticnet*. Celle-ci garde toutes les variables en entrée, mais les pénalisent lors de l'apprentissage.

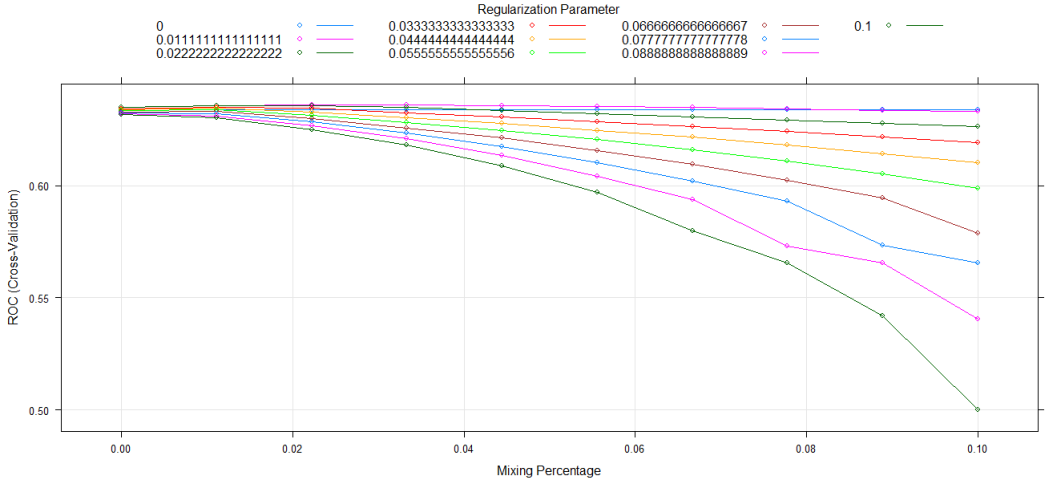


FIGURE 2.1: Stabilisation des paramètres glmnet

Nous pouvons voir la stabilisation des paramètres sur la Figure 2.1. Avec ce modèle (nommé glmnet) je suis monté à **66.08%** d'AUC avec $\lambda = 0.011$ et $\alpha = 0.022$.

2.2 RandomForest

Après avoir testé le modèle le plus basique, j'ai commencé par utiliser les arbres de décision, et plus précisément le randomforest. Celui-ci est connu pour être un des modèles les plus utilisés dans le monde industriel pour avoir les meilleures prédictions tout en étant un modèle simple. Celui-ci est aussi connu pour ne jamais overfitter grâce à la façon dont il est construit (une sélection de lignes Booststrap, et une sélection de colonnes aléatoire). J'ai donc commencé par lancer le premier modèle (nommé random.forest), puis j'ai vu avec la courbe d'erreur que 100-200 arbres suffiront avec une AUC de **61.58%**. J'ai commencé par régler le paramètre Mtry avant tout, et j'ai eu plusieurs résultats qui alternaient entre 3,5 et 7. Puis j'ai réglé les paramètres Mtry avec le nombre d'arbres pour avoir les meilleures performances (modèle nommé random.forest.tuned), mais les résultats n'ont pas été satisfaisant (AUC de **59,33%**). Les meilleurs paramètres trouvés étaient $Mtry$=7 avec 200 arbres. Je suis donc passé sur un autre type d'arbre : le Gradient Boosting.

Tentative	eta	colsample_bytree	max_depth
1	0.1	0.2	4
2	0.1	0.2	3
3	0.125	0.225	3
4	0.125	0.21	3

FIGURE 2.2: Différentes étapes du GridSarch de Xgboost

2.3 Gradient Boosting

Le principe du gradient boosting est de créer un arbre simple (qui peut être très faible), et de le corriger au fur et à mesure en donnant plus de poids aux points mal étiqueté à l'itération précédente. J'ai commencé par un modèle (nommé gbm) en cross-validation en utilisant le package *caret*, et les résultats étaient plutôt satisfaisants par rapport aux modèles précédents. J'arrive à une AUC de **67,2%** avec les réglages suivants :

- Shrinkage = 0.1
- minobsinnode=10
- interaction.depth=1
- ntrees=150

Puis en "jouant" un peu avec les paramètres (en utilisant la *GridSearch*) , et toujours en restant en cross-validation, ce nouveau modèle (nommé gbmFit) arrive à une AUC de **68,77%** grâce aux paramètres suivants :

- Shrinkage = 0.15
- minobsinnode=10
- interaction.depth=1
- ntrees=600

Et finalement, en augmentant légèrement le nombre d'arbres à 800 ce modèle (nommé gbmFit.second) m'a permis de m'approcher au maximum du cap des 70%, plus précisément une AUC de **69,05%**. J'aurais pu augmenter la taille des arbres une troisième fois, vu que les autres paramètres ont convergé vers une valeur, mais j'ai voulu m'éloigner du risque de l'overfit.

2.4 EXtreme Gradient Boosting

Après avoir obtenu un meilleur score avec le gradient boosting, je me suis mis a travailler avec l'eXtreme Gradient Boosting aussi nommé Xgboost. Celui-ci suit la même logique que le *gbm*, en ayant en plus une pénalisation. D'habitude, cet algorithme est utilisé dans les compétitions pour avoir les meilleurs scores (aux pourcentages près).

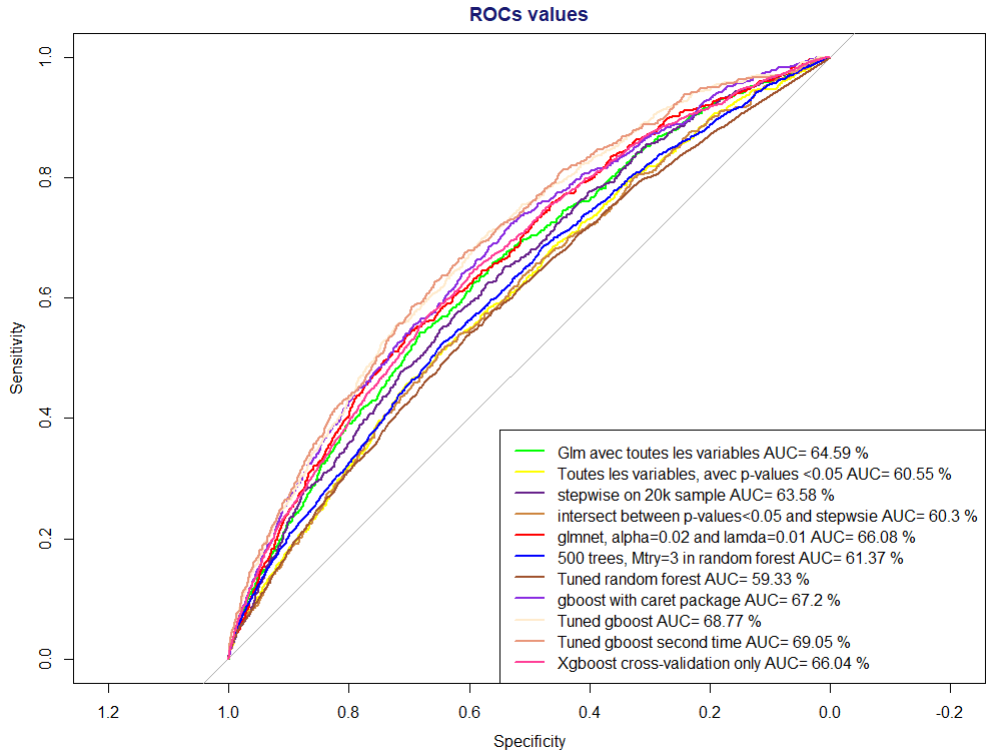


FIGURE 2.3: La courbe ROC de tous les modèles créés

J'ai lancé un modèle par défaut comme d'habitude mais celui-ci n'était pas très performant. J'ai donc débuté le *Tunning parameters* du modèle. Premièrement, j'ai cherché par trouver la meilleure combinaison entre *eta*, *colsample_bytree* et *max_depth*. Pour trouver le meilleur modèle, j'ai commencé par leur attribué des valeurs aléatoires, et les corriger jusqu'à converger vers la meilleure combinaison entre eux. À savoir le tableau suivant 2.2 présente les différentes étapes par lesquelles la gridSearch est passée. Les calculs ont, en plus, été fait en cross-validation.

Après avoir trouvé la meilleure combinaison, j'ai commencé à travailler *gamma* tout seul vu son importance dans ce modèle.

À la fin de cette étape, j'ai retenu les paramètres suivants (modèle nommé fit.xgb.cv) pour arriver à 66.04% d'AUC sur la cross-validation (pas ma base de test).

— eta = 0.125
— subsample = 0.9
— colsample_bytree = 0.21

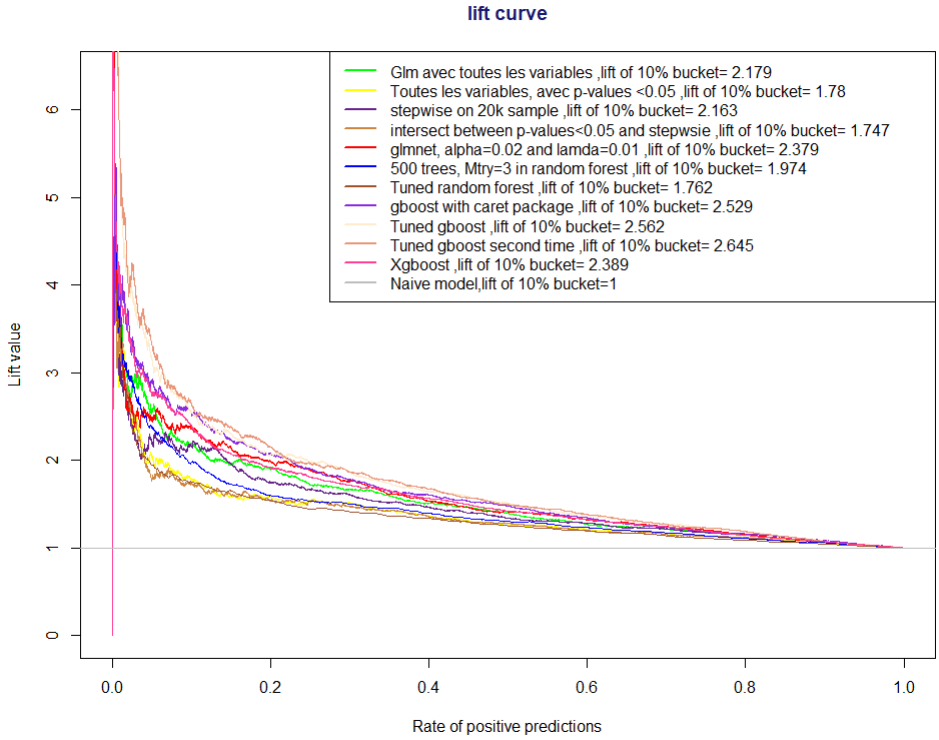


FIGURE 2.4: La courbe lift, avec la valeur du premier bucket (10%) de tous les modèles créés

— max_depth = 3
— gamma = 4

J'ai dû entrainer un nouveau modèle (nommé xgb) pour pouvoir obtenir des prédictions, car l'ancien apprentissage (fit.xgb.cv) était une cross-validation que ne retourne aucun modèle. J'arrive à une AUC de **66,3%** sur mes données de test. Je m'attendais à dépasser le Gradient Boosting dans cette étape et avoir un peu plus d'AUC mais ce n'était pas réussi.
J'ai essayé de lancer un modèle SVM. Le premire modèle (non réglé) n'était pas très performant comme d'habitude (envrion 57% d'AUC) J'ai donc commencé les réglages du SVM, mais une cross-validation avec une GridSearch a coûté plus de trois jours d'exécution (et ce n'était pas encore fini). J'ai donc finit par abandonner ce modèle.

2.5 Évaluation

La performance des modèles a été évaluées grâce aux indicateurs *AUC* Area Under of Curve et le Lift. Le premier indicateur trace une courbe avec en abscisse et en ordonnée. Avoir une AUC de 1 est le *TRUTH* modèle, que l'on ne peut jamais atteindre, tandis qu'une AUC de 0.5 est une modélisation du hasard. L'ensemble des AUC trouvées pour mes modèles sont représenté dans la figure suivante 2.3.

Le lift est également un indicateur de performance mais différent de l'AUC. Il permet de viser le maximum de personnes intéressées. Par exemple, si nous avons 20% de l'ensemble des personnes de notre base de données qui veulent acheter notre produit. Avec la courbe lift, nous divisons nos prédictions par déciles pour ensuite calculer le nombre de personnes intéressées (par décile toujours). Avec cette valeur, nous pouvons décider de cibler le premier bucket seulement, c'est-à-dire le premier décile et multiplier nos chances de tomber sur les personnes intéressées de 2 voire 3 fois (c'est la valeur du lift). Dans le projet, j'ai calculé le premier bucket du lift et l'ai utilisé comme indicateur de performance à côté de l'AUC. L'ensemble des premiers buckets trouvés sur mes modèles sont représentés sur la Figure 2.4. Le modèle *Xgboost* multiplie nos chances de tomber sur un client appétant de **2,389** fois. Nous remarquons que la courbe atteint parfois un lift de 6, pourtant le meilleur lift ne dépasse pas les 3. Ceci est dû au nombre de valeurs prises par les x : il existe environ 17 000 valeurs et pas que 10. Pour tracer la courbe, je laisse tous ces groupes pour une belle courbe mais pour calculer la valeur du premier bucket, je calcule les premiers 10% de la population.

Importance de variables

Pour trouver les meilleurs variables j'ai utilisé plusieurs techniques différentes :

1. stepwise
2. *earth*
3. la fonction d'importance du *randomforest*
4. la fonction d'importance de *caret*
5. les informations values
6. *Boruta*

3.1 Stepwise

Cette méthode de sélection de variable est une branche du modèle *glm* qui permet de sélectionner les meilleures variables grâce à leurs valeurs de p-values. Cette méthode ajoute et supprime des variables pour arriver à la meilleure combinaison de variable possible. Dans le projet, et vu la complexité de ce calcul, j'ai pris un échantillon de 20 000 données pour appliquer le stepwise. Notons que cette sélection de variable est faite avec les variables *dummies*. Voici les variables trouvées durant de cette étape : $age\_prospect$, var_5, var_6, var_7, var_10, var_13, var_14, var_15, var_16, var_18, var_21, $sexe.F$, $var\_23.B$, $var\_23.C$, $var\_24.B$, $var\_27.A$, $var\_28.B$, $var\_32.E$, $var\_32.H$, $var\_32.S$, $var\_33.F$, $var\_33.J$, $var\_31.K$, $var\_33.I$, $var\_33.R$.
Nous pouvons voir leurs impacts sur la prédiction de target, en comparant les AIC provenant des modèles glm1 et glm2 créés auparavant, et l'AIC que le stepwise a trouvé sur l'échantillon. Un AIC ne peut être interprété seul comme indicateur de performance, mais il est utilisé pour comparer deux modèles : celui qui produit un AIC inférieur est le meilleur des deux. Nous pouvons facilement remarquer une baisse énorme de l'AIC : 15219 pour *glm1*, 15322 pour *glm2* et 4368 pour le stepwise (de l'échantillon).
Aussi, nous remarquons que le modèle créé à partir des coefficients du stepwise a une AIC supérieur du stepwise de l'échantillon, car justement l'échantillon fait perdre des informations utiles. Pour avoir la meilleure combinaison (et donc le meilleur modèle), on aurait dû utiliser toute la base de données, mais on aura eu beaucoup de problèmes de ressource notamment le temps de calcul.

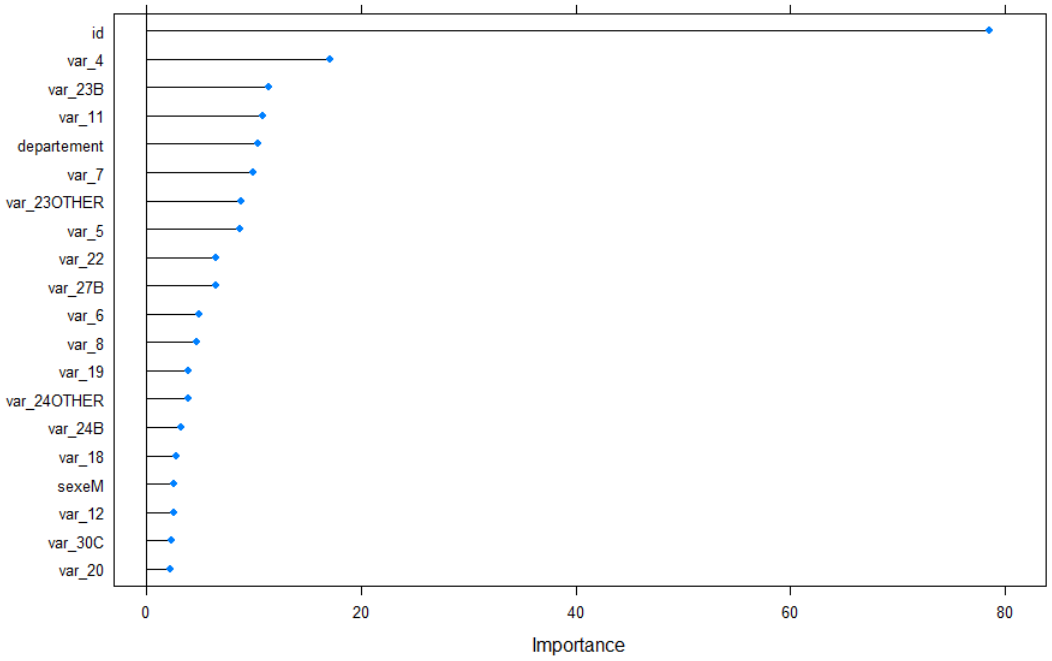


FIGURE 3.1: Les importances des variables du meilleur modèle gbm

3.2 Fonction d'importance

Dans cette partie, j'ai utilisé les fonctions d'importances disponibles pour les randomforest mais également pour les modèles créés avec *caret*. Je vais présenter ici la fonction d'importance du meilleur modèle *gbm* trouvé, mais les fonctions d'importance de chaque modèle (randomforest ou caret sont disponibles dans les plots envoyés).
Ce dernier modèle donne une très grande importance à la variable *id*, qui est d'ailleurs très importante dans les autres modèles gbm également. Puis nous trouvons *var_4* à la seconde place et *var_11*, *departement* et *var_7* ayant à peu près la même importance. Mais la variable *id* est la plus importante de toutes, avec une importance de 78,49 contre 17,01 pour la seconde variable *var_4*.

3.3 Informations Values

Cette technique permet de trouver les variables **qualitative** les plus importantes du modèle. Grâce au package *Informations Value* disponible sous

VARS	IV	STRENGTH
var_23	0.08153474	Somewhat Predictive
var_26	0.04463900	Somewhat Predictive
var_25	0.04398198	Somewhat Predictive
var_24	0.03251516	Somewhat Predictive
var_32	0.02698974	Not Predictive
var_31	0.02665311	Not Predictive
var_28	0.02488371	Not Predictive
var_33	0.02436184	Not Predictive
var_30	0.02325128	Not Predictive
var_27	0.02293074	Not Predictive
var_29	0.01160235	Not Predictive
sexe	0.01118474	Not Predictive

FIGURE 3.2: Résultats de l'Information Value par rapport à target

R, nous pouvons calculer l'information Value IV basée sur le Weight of Evidence WOE. WOE et IV sont calculés de la façon suivante :
Supposons PR = pourcentage des bons (des 1) et PM = pourcentage des mauvais (des 0), on a :

$$WOE = \ln(\frac{PR}{PM})$$

$$IV = (PR - PM) \times WOE$$

L'IV est interprétée de cette façon :

- moins de 0,03, cette variable n'a pratiquement aucune influence sur target.
- 0,03 à 0,1, cette variable a une liaison faible avec le target.
- 0,1 à 0,3, cette variable a une liaison moyenne avec le target.
- 0,3 à 0,5, cette variable a une forte liaison avec le target.
- supérieur à 0.5, suspicieux, étonnamment liée à target.

Parfois la première règle est modifiée : ses valeurs sont inférieurs à 0,02 et pas 0,03. En suivant cette règle, nous avons trouvé 4 variables qualitatives faiblement liées à target, et 8 variables non liées à target. Les résultats sont présentés dans le tableau 3.2

3.4 Boruta

La technique de Boruta est celle qui a pris le plus de temps à s'exécuter. Après 99 itérations durant *5.678539 heures*, le modèle *boruta* a rejeté deux

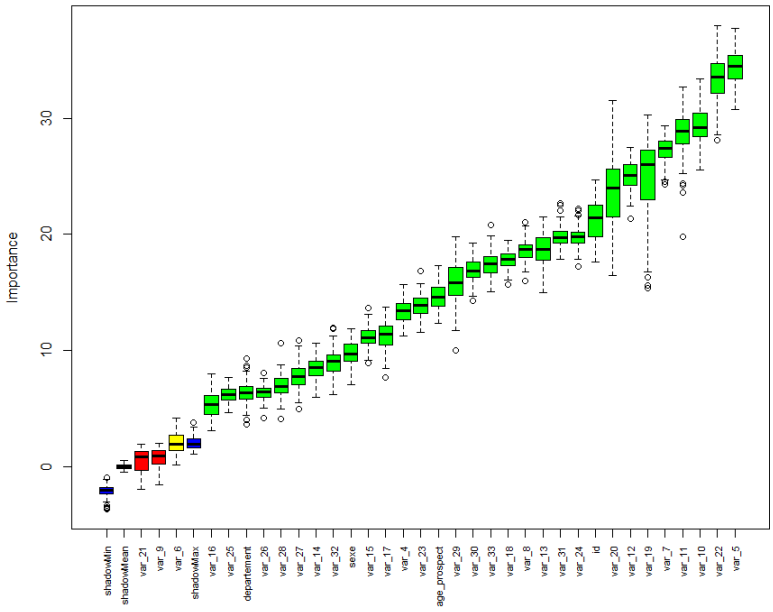


FIGURE 3.3: Les importances des variables expliquée par Boruta

variables et a gardé toutes les autres. Les variables rejetées sont var_9 et var_21, alors que var_6 *boruta* n'a pas pu conclure vis-à-vis de son importance. Aussi, on trouve ici que var_5 et var_22 sont -d'après lui- les plus importantes variables de la base comme le montre la figure 3.3. Les détails de la méthode et d'autres informations comme l'historique de l'importance de chaque variable sont disponibles dans l'objet R nommé *boruta.train*.

Les variables retenues avec cette méthode sont alors les suivantes : *departement*, $age\_prospect$, var_4, var_5, var_7, var_8, var_10, var_11, var_12, var_13, var_14, var_15, var_16, var_17, var_18, var_19, var_20, var_22, id, $sexe$, var_23, var_24, var_25, var_26, var_27, var_28, var_29, var_30, var_31, var_32, var_33.

J'ai également travaillé avec le package *earth* qui permet de trouver les meilleurs variables en cross-validation, et en minimisant la fonction RSS du modèle. Malheureusement je n'ai pas pu exploiter toutes ces fonctions pour arriver à la meilleure sélection de variable. Néanmoins, avec un modèle simple, j'ai trouvé 4 variables indispensable au modèle : var_23B, var_11, var_4, var_5 citées par ordre d'importance. Je n'ai pas pu trouver leurs impacts sur la target.

Pour conclure la partie des importances des variables, j'ai utilisé plusieurs modèles et algorithmes pour trouver les meilleures variables, mais aucune technique ne sera à 100% d'accord avec une autre. Par contre, nous trouvons des variables que se répètent souvent dans les variables importantes comme par exemple $var_2 3$ et plus précisément son facteur **B**. Nous pourrons conclure que ces variables sont alors plus importantes que les autres.