Université de Lille

Faculté des Sciences et Technologies Département de Mathématiques

---

Master : Ingénierie, Statistique et Numérique - Data Science

Projet Méthodes d'apprentissage

Classifier si un logiciel est malware ou goodware

Réalisé par :

Youssouf Roki-Dine et Victor Vannobel

Année universitaire : 2023-2024

Table des matières

I Introduction

L'objectif de ce projet consiste à développer un classifieur capable de différencier un logiciel si il est **malware** ou **goodware** en utilisant une base de données comprenant 4465 individus évalués selon 241 variables. Au cours de cette étude, nous évaluerons et comparerons les performances de divers algorithmes.

Après une première inspection des données, nous constatons qu'un individu ne présente aucune information (NaN) et nous décidons de le supprimer de la base de données. De plus, de nombreux individus semblent être dupliqués au sein de la base de données, nous ne décidons cependant pas de les supprimer. Ainsi, la base de données finale comprend 4464 individus évalués selon 241 variables.
Pour amorcer cette étude, nous entreprendrons une analyse statistique approfondie des données. Ensuite, nous mettrons en œuvre différents algorithmes de classification. Enfin, nous chercherons à comparer les performances et les résultats des algorithmes pour trouver ceux qui sont les mieux adaptés pour prédire les labels.

II Étude statistique préliminaire

L'étude statistique sert à mettre en évidence les caractéristiques des variables qui pourraient compromettre l'efficacité des algorithmes. Les algorithmes peuvent être sensibles aux données extrêmes, à la multicolinéarité et à la proportion de chaque label. Nous examinerons tour à tour chacune de ces caractéristiques.

II.1 Statistiques descriptives

On constate que la base de données comporte deux classes qui seront symbolisées par un 1 ou par un 0 lorsque le logiciel est **malware** ou **goodware**. On s'intèresse à la proportion de chaque classe : 80% de **malwares** contre 20% de **goodwares**, les classes sont déséquilibrées (classe **malware** largement majoritaire). Une étude sera réalisée dans la suite du rapport après équilibrage des classes par différentes méthodes.

Il est intéressent de voir quelles variables pourraient avoir un effet pour déterminer si un logiciel est **malware** ou non. Prenons la variable "RECEIVE_BOOT_COMPLETED" : on constate d'après la figure 1 que 98% des **malwares** ont cette variable contre 29% des **goodwares**.

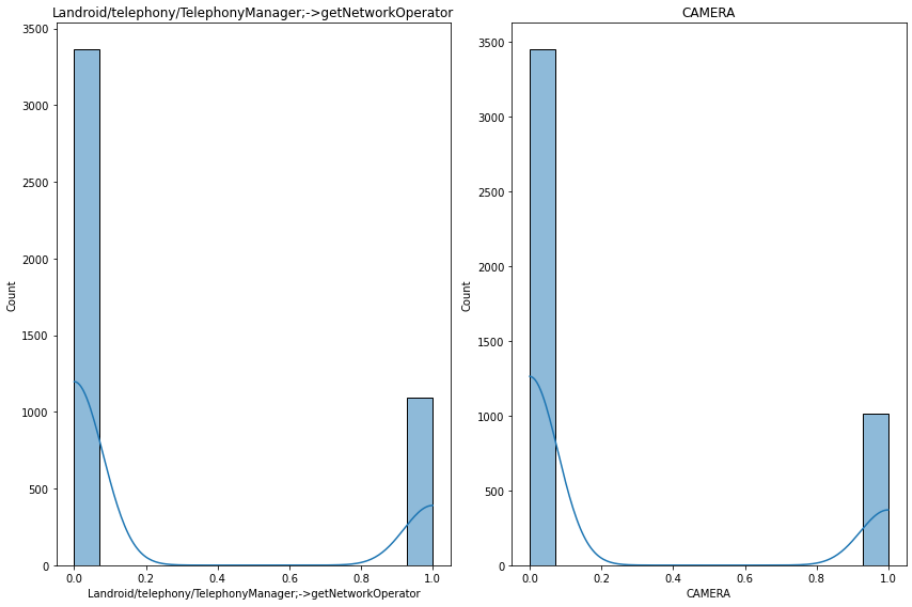
```
Moyenne RECEIVE_BOOT_COMPLETD malware : 0.982, goodware : 0.294
```

(Figure 1) Ratios des classes pour la variable "RECEIVE_BOOT_COMPLETED".

On regarde aussi les variables pour lesquelles les variances par classes sont significativement différentes. Pour cela on compare les intervalles de confiance et on trouve 181 variables. Il y'a beaucoup de variables qui permettent de séparer les classes **malwares** et **goodwares**, on s'attend à avoir de bonnes erreurs de prédiction.

II.2 Valeurs extrêmes

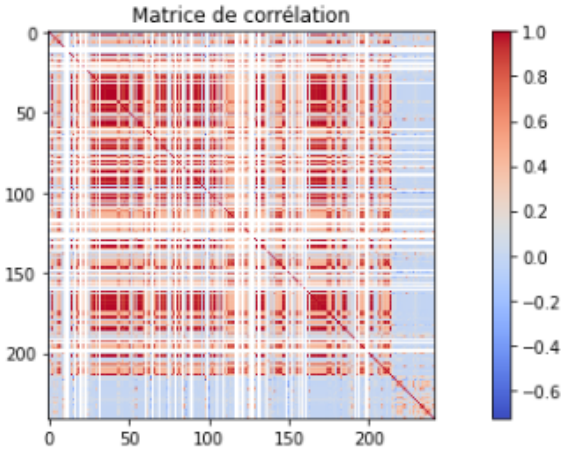
Les données extrêmes peuvent être à l'origine d'une instabilité des résultats pour les algorithmes dont les coefficients s'expriment comme une fonction des données (notamment une moyenne qui est très sensible aux valeurs extrêmes). En fonction de si elles sont présentes dans l'échantillon d'entraînement ou non, la valeur des coefficients peut fortement varier.
Pour les détecter on représente les histogrammes des variables. La distribution de deux d'entre elles se démarque. On représente les histogrammes de ces deux variables dans la figure 2.



(Figure 2) Histogrammes des variables.

Les deux variables comportent de nombreuses valeurs "extrêmes". Mais celle-ci sont tellement nombreuses et proches qu'elles ne représentent potentiellement pas un danger pour la stabilité de nos résultats : la proportion de valeurs élevées devrait être similaire et assez stable dans les échantillons d'entraînement et de test.

II.3 Régression gaussienne



(Figure 3) Visualisation de la corrélation des variables.

La matrice de corrélation (figure 3) montre que certaines variables sont très corrélées entre elles. Les modèles comportant des inversions de matrices tels que la régression gaussienne sont donc à éviter. Les variables fortement corrélées peuvent entrainer des estimations instables pour nos futurs modèles.

II.4 Cardinal des classes

Dans notre cas, nous constatons un déséquilibre significatif, avec seulement 20% des échantillons de classe **goodware**. Cette disparité peut entrainer une minimisation du taux d'erreur de classification associé au logiciel **goodware**. Ainsi, pour minimiser les erreurs de classification de **malwares** en tant que **goodwares**, le modèle tendrait à prédire plus de **goodwares** pour s'assurer de ne pas manquer les **malwares**, même au détriment de classer certains **goodwares** comme **malwares**.

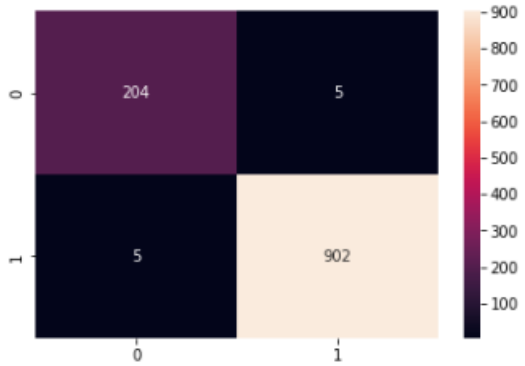
III Algorithmes de classification

À partir de la base de données originale, nous créons une base de données d'apprentissage ainsi qu'une base de données de test de dimension (sur l'espace des individus) égale à 25% de la dimension originale soit 1116 individus de test. Nous utiliserons la graine 123 pour ne pas que nos résultats changent à cause des processus aléatoires. Ces bases de données seront utilisées par la suite pour entrainer et tester les capacités prédictives des modèles.

III.1 Régression logistique

Dans la première section, nous avons observé l'absence de données extrêmes. Par conséquent, l'application d'une classification logistique est envisageable, offrant ainsi l'avantage de ne nécessiter aucune optimisation d'hyper paramètres si ce n'est le paramètre de régularisation C que nous prendrons à 40 et de permettre une interprétation aisée des résultats grâce aux odd-ratios.

Une fois le modèle entrainé, on lit une erreur de 0.63% sur les données d'entraînement ainsi qu'une erreur de 0.89% sur les données de test. Regardons la matrice de confusion pour les données de test. Les lignes de la matrice de confusion représentent les labels réels et les colonnes représentent les prédictions. Un modèle qui ne se trompe pas a donc une matrice de confusion diagonale. Ainsi d'après la figure 4, 5 logiciels ont été classés **malwares** alors qu'ils étaient **goodwares** (faux positifs) et 5 logiciels ont été classés **goodwares** alors qu'ils étaient **malwares** (faux négatifs).



(Figure 4) Matrice de confusion pour la régression logistique (données de test).

Stabilité et interprétation des résultats

Dans cette sous section, nous recommençons l'apprentissage du modèle par régression linéaire 100 fois avec différentes bases de données d'entraînement et de test afin de construire un intervalle de confiance empirique à 95% pour l'erreur sur la prédiction. On obtient après la procédure une erreur moyenne sur la prédiction de 1.5%, une variance de 1.7e-05 et un intervalle de confiance IC(95%) = [0.70516344, 2.30558925].

La régression logisitique offre la possibilité d'avoir facilement accès aux variables influentes. Intéressons nous donc aux variables qui ont un effet pour déterminer si un logiciel a une plus forte probabilité ou non d'être un **malware** grâce aux odd-ratios (odd-ratio»1 : probabilité forte, odd-ratio $\sim$ 1 : peu influent, odd-ratio«1 : probabilité faible). On décide arbitrairement d'afficher les odd-ratios supérieurs à 6 et inférieurs à 0.2 (figure 5) :

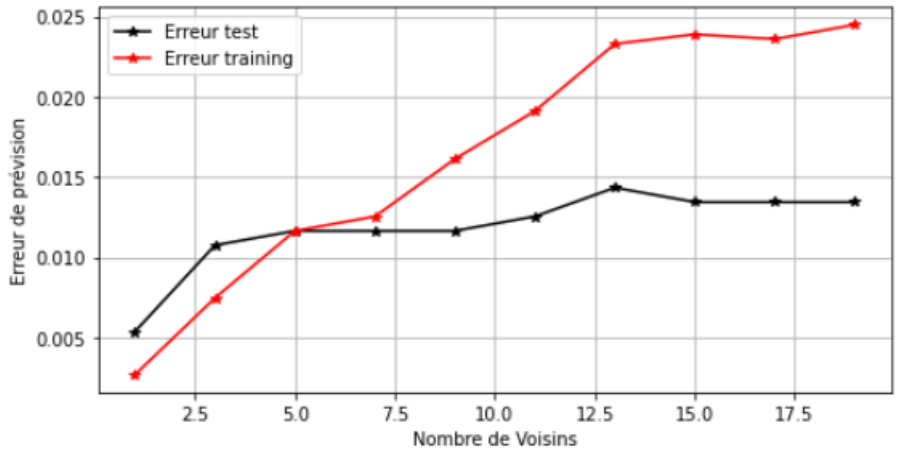
```
odd>6
KILL_BACKGROUND_PROCESSES
RECEIVE_BOOT_COMPLETED
RECEIVE_SMS
SYSTEM_ALERT_WINDOW

odd<0.2
VIBRATE
WRITE_SETTINGS
Ljava/net/URL;->openConnection
```

(Figure 5) Quelques odd-ratios.

Vérifions avec notre base de données pour la variable "KILL_BACKGROUND_PROCESSES" : 61.4 % des **malwares** la possèdent contre 6.7% des **goodwares**.
De même pour la variable "VIBRATE", 14.5% des **malwares** la possèdent contre 41.7% des **goodwares**.

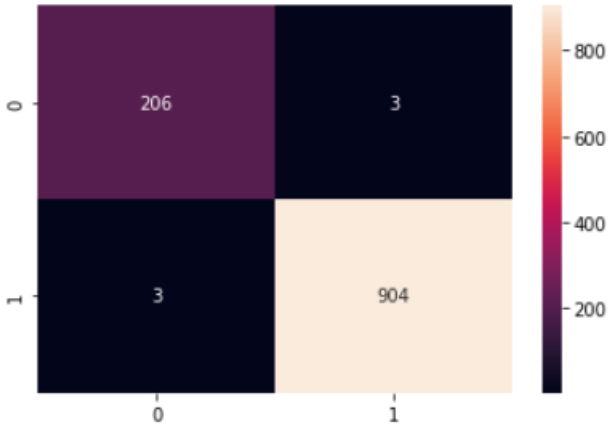
III.2 K plus proches voisins (KPP)

La distribution des classes est asymétrique. La classe majoritaire est statistiquement plus fréquente parmi les K plus proches voisins par définition. On souhaite K impair pour éviter les problèmes d'indécision étant donné que nous n'avons que deux classes. Nous cherchons le nombre K de voisins minimisant l'erreur de test :



(Figure 6) Quelques odd-ratios.

On lit sur la figure 6 que le nombre de voisins qui minimise l'erreur de test est 1.
Plus précisemment, il y'a une erreur de 0.27% sur les données d'entraînement ainsi qu'une erreur de 0.54% sur les données de test.
Voyons la matrice de confusion sur les données de test :



(Figure 7) Matrice de confusion pour les K plus proches voisins (K=1, données de test).

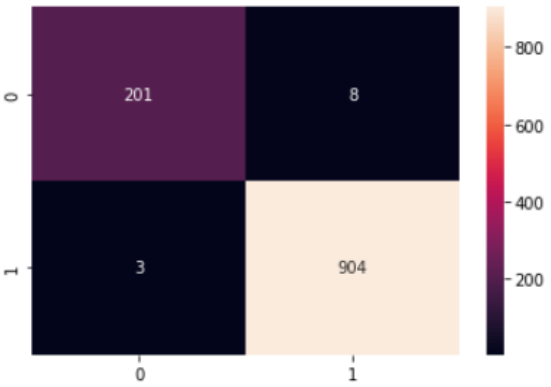
On lit 3 faux négatifs et 3 faux positifs (figure 7).

Il est possible de rendre automatique la recherche du K optimal grâce à la fonction **GridSearchCV**. La valeur de CV en argument détermine la statégie de séparation par validation croisée et il n'est pas garanti que la fonction renvoie les hyper paramètres optimaux pour avoir l'erreur de prédiction la plus faible mais plutôt ceux qui maximisent les performances (pas seulement l'erreur de prédiction) sur les données d'entraînement fournies. Il va être intéressant d'utiliser cette fonction pour des modèles plus complexes comme le suivant par exemple.

III.3 Arbres décisionnels

Nous allons utiliser la fonction **GridSearchCV** pour déterminer la profondeur optimale de l'arbre : la fonction renvoie 18.

On trouve une erreur sur les données d'entraînement de 0.27% contre une erreur de 0.99% sur les données de test.

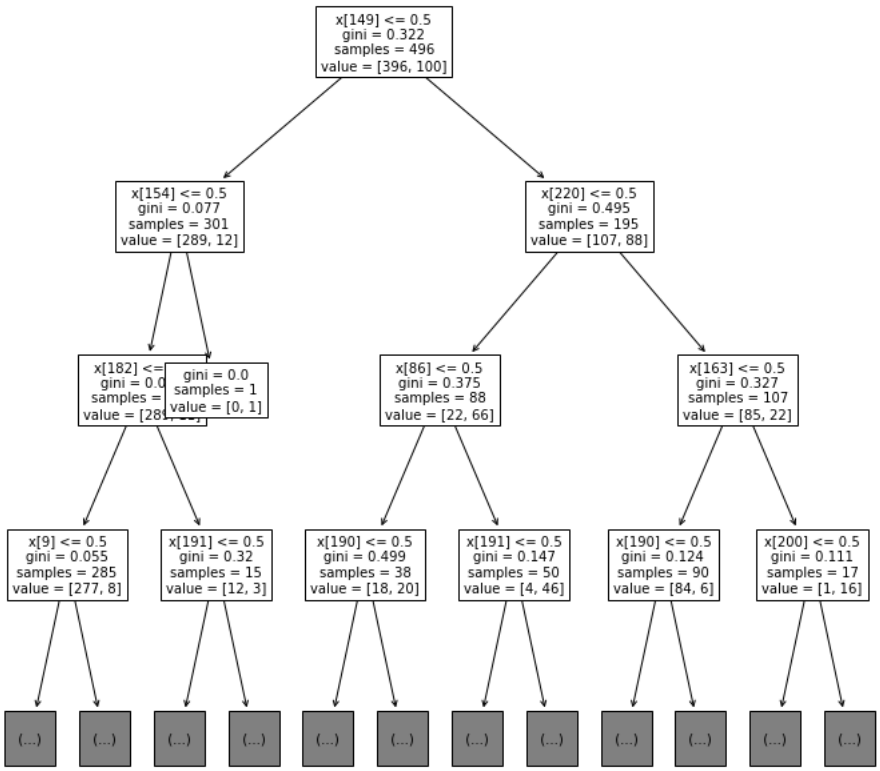


(Figure 8) Matrice de confusion pour notre modèle d'arbre décisionnel (données de test).

On lit 3 faux négatifs et 8 faux positifs (figure 8). L'erreur de prédiction est plus élevée que précédemment.

Interprétation des résultats

Pour savoir quelles variables ont le plus compté dans l'algorithme, on représente un arbre de classification évalué avec les mêmes paramètres. La figure 9 nous montre le schéma de séparation d'un arbre.



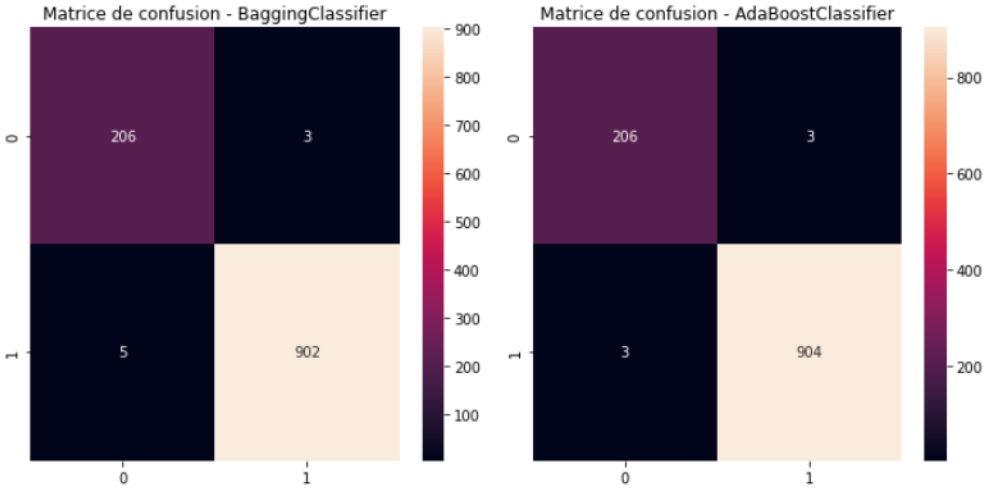
(Figure 9) Un arbre de décision.

Les variables qui ressortent en premier sont celles qui discriminent le mieux nos individus. Si on s'en réfère aux premiers nœuds pour cet arbre, les variables les plus importantes pour la classification sont les variables : "RECEIVE_BOOT_COMPLETED", "RECEIVE_WAP_PUSH", "GET_TASKS", "Ljava/net/URL;->openConnection", "SYSTEM_ALERT_WINDOW", "SEND_SMS" ...

Bagging et boosting

Nous allons ici utiliser les techniques de bagging et boosting pour améliorer les modèles d'arbres décisionnels. Les hyper paramètres ont été optimisés avec la fonction **GridSear-**

chCV, pour le bagging on prendra 18 estimateurs et pour le boosting 100 estimateur ainsi qu'un learnig rate de 0.5. On affiche ci-dessous les résultats obtenus sous forme de matrice de confusion :



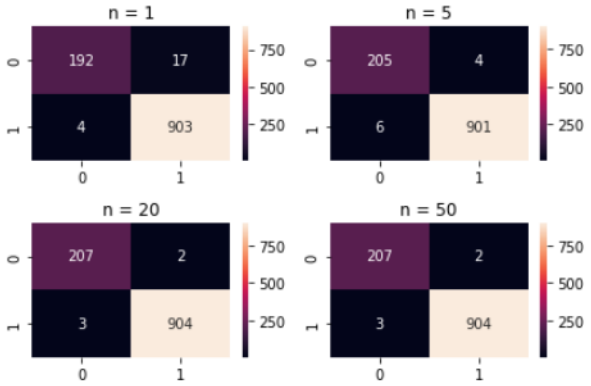
(Figure 10) Matrices de confusion pour bagging et boosting (données de test).

D'après la figure 10, on constate sur les données de test une erreur pour le bagging de 0.7% et pour le boosting de 0.5%. Les erreurs sur les données d'entraînement sont les suivantes : 0.3% et 0.27%.
Ces deux techniques permettent d'améliorer les résultats prédictifs du modèle d'arbre décisionnel.

Voyons ce qu'il se passe en combinant les arbres grâce aux forêts aléatoires, pour réduire le facteur sur-apprentissage.

III.4 Forêts aléatoires

En général, les forêts aléatoires utilisent déjà le concept de bagging pour construire un ensemble d'arbres de décision plus robuste. On ne fera pas de bagging sur les modèles de forêts aléatoires et faire du boosting pourrait ne pas être bénéfique ou être redondant. Pour revenir à la corrélation des variables, cela diminue la capacité de généralisation du modèle et pourrait entrainer une baisse de performance car les arbres de la forêt pourraient être moins capables de capturer différentes nuances et relations dans les données.
Nous allons construire des forêts aléatoires prenant comme nombre d'arbres les éléments de {1,5,20,50}.



(Figure 11) Matrices de confusion pour différentes forêts aléatoires (données de test).

On trouve réspectivement des erreurs sur les données de test de 1.9%, 0.89%, 0.44%, 0.44% (figure 11). Les erreurs sur l'entraînement : 0.75%, 0.33%, 0.27%, 0.27%.
Il est important de noter qu'à partir d'un certain nombre d'arbres, continuer à augmenter le nombre d'arbres ne fera plus nécessairement baisser l'erreur de prédiction.

Stabilité des résultats

Nous calculons un intervalle de confiance sur l'erreur de prédiction à 95 % pour le modèle de forêts aléatoires : IC(95%) =[0.2413716, 0.74429148].

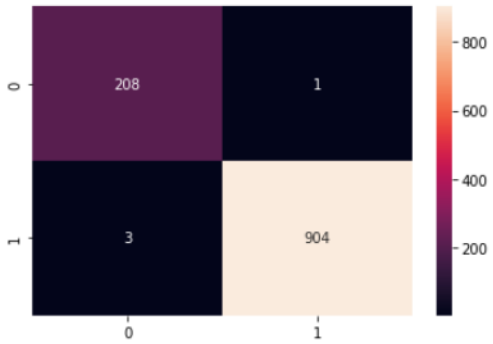
III.5 Support Vecteur Machine (SVM)

La SVM est une méthode consistant à chercher une dimension supérieure dans laquelle nos données seraient séparables par une droite linéaire. Comme la forêt aléatoire, elle est insensible aux données extrêmes mais la séparation pourrait être plus compliquée à cause d'une forte corrélation entre les variables. De plus, le modèle nécessite de choisir de nombreux hyper paramètres, ce qui peut être difficile et long à mettre en œuvre. Enfin, les résultats sont difficilement interprétables.

Choix des paramètres et prévisions

La base de données possédant de grandes dimensions, l'utilisation de la SVM pour traiter les données pourrait être efficace. On considère un modèle de SVM dont les hyper paramètres C et gamma sont optimisés par la fonction **GridSearchCV**.
La régularisation est proportionnelle à l'inverse de C, la valeur de C donnée est 5. Le paramètre tol fait référence à une condition d'arrêt pour la résolution du problème d'optimisation, si la différence entre deux itérations de la fonction d'optimisation est inférieure à tol, l'algorithme considère qu'il a convergé. On prendra tol par défaut. Enfin le paramètre

gamma définit l'importance dans l'apprentissage de s'adapter de manière plus ou moins précise aux données d'entraînement. On prendra une valeur de 0.2. Voyons les résultats :



(Figure 12) Matrice de confusion pour notre modèle de SVM (données de test).

On constate une erreur de 0.27% sur les données d'entraînement et de 0.36% sur les données de test (figure 12). On constate que l'erreur de prédiction la plus basse est obtenue avec le modèle de SVM.

Stabilité des résultats

Nous calculons un intervalle de confiance sur l'erreur de prédiction à 95 % pour le modèle de SVM : IC(95%) =[0.29186933, 0.83716292].

III.6 Récapitulatif

Erreurs

	Error_train	Error_test	FP	FN
Logis	0,63 %	0,89 %	5	5
KPP	0,27 %	0,54 %	3	3
Arbre	0,27 %	0,99 %	8	3
Bagging	0,30 %	0,70 %	3	5
Boosting	0,27 %	0,50 %	3	3
Forêts n=50	0,27 %	0,44 %	2	3
SVM	0,27 %	0,36 %	1	3

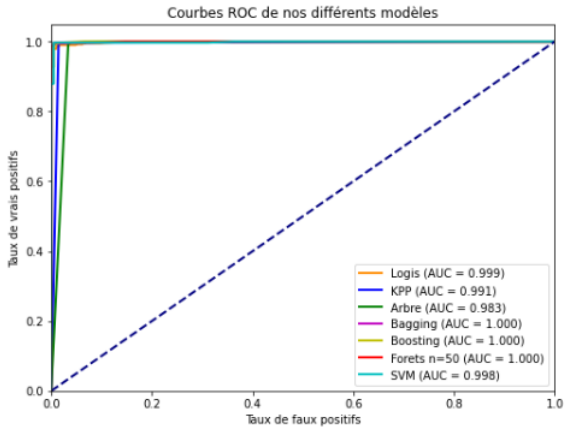
(Figure 13) Erreurs d'entraînement et de test pour les différents modèles.

On constate que les modèles qui donnent les meilleures prédictions sont les forêts aléatoires et la SVM (figure 13).

Courbes ROC

Nous représentons graphiquement les caractéristiques opérationnelles du récepteur (ROC) de chaque algorithme, où le taux de vrais positifs (TVP) est tracé en fonction du taux de vrais négatifs (TVN). Le TVP représente le nombre de vrais positifs par rapport au nombre total de positifs (vrais et faux), et de même pour le TVN. L'aire sous

la courbe ROC (AUC) correspond à la moyenne du taux de vrais positifs et du taux de vrais négatifs (AUC = (TVP + TVN)/2). Les courbes ROC de chaque algorithme sont présentées dans la figure 14.



(Figure 14) Courbes ROC de nos différents modèles.

On peut considérer qu'un modèle prédit suffisamment bien lorsque son AUC est supérieure à 0.90. En l'occurence, tous nos modèles excepté le modèle d'abre décisionnel ont une AUC supérieure à 0.99. On notera que les modèles de forêts aléatoires, SVM et régression logistique offrent les meilleures performances en termes de prévisions (AUC de 1, 0.998 et 0.999). On constate que la régression logistique a une AUC proche de 1 tout en ayant eu une des pires erreurs de prédiction (relativement aux autres modèles), cela peut s'expliquer par le déséquilibre des classes ou encore que pour calculer l'AUC, l'erreur de prédiction n'est pas le seul critère en jeu.

IV Rééquilibrage des classes

On souhaite désormais rééquilibrer les classes. Pour ce faire, nous allons utiliser les fonctions NearMiss et SMOTENC. Une fois les nouvelles données d'entraînement obtenues, on constate qu'il y'a 50% de **malwares** et 50% de **goodwares**.

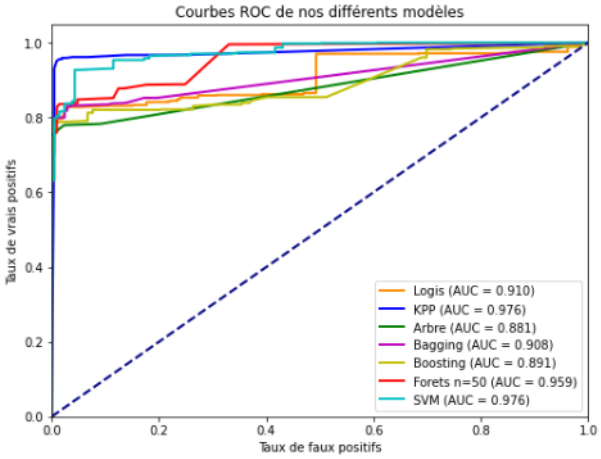
IV.1 NearMiss

La base de données d'entraînement contient 1380 individus, ce qui peut être insuffisant pour entrainer correctement certains modèles.
On affiche directement les erreurs sur les données d'entraînement et de test pour les différents logiciels sous forme de tableau :

NearMiss	Error_train	Error_test	FP	FN
Logis	0,87 %	15,68 %	5	170
KPP	3,67 %	3,85 %	6	37
Arbre	0,80 %	18,40 %	5	200
Bagging	0,65 %	16,67 %	4	182
Boosting	0,65 %	17,47 %	2	193
Forêts n=50	0,65 %	19,70 %	2	218
SVM	0,65 %	23,20 %	1	258

(Figure 15) Erreurs d'entraînement et de test pour les différents modèles (NearMiss).

Sauf pour les KPP, le taux d'erreur de prédiction pour chaque modèle est assez important (figure 15). En regardant en détail les erreurs commises, le nombre de faux négatifs est beaucoup plus élevé que le nombre de faux positifs alors que l'on aimerait l'inverse. De plus, la SVM qui précédemment était le meilleur modèle en terme d'erreur de prédiction est désormais le pire. Regardons les courbes ROC :



(Figure 16) Courbes ROC des différents modèles (NearMiss).

La plupart des modèles ont une AUC supérieure à 0.9 (figure 16). Cependant, le calcul de l'AUC ne prend pas en compte les faux positifs/négatifs et pour la plupart des modèles, il y'a trop de faux négatifs. Les résultats avec NearMiss ne sont pas bons.

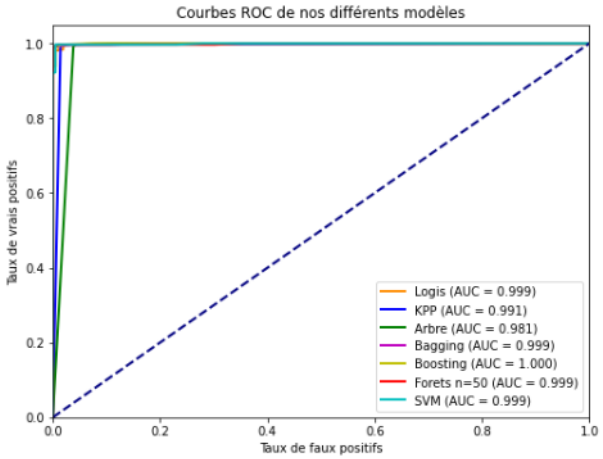
IV.2 SMOTENC

La base de données d'entraînement contient 5315 individus. Affichons les erreurs pour chaque modèle :

SMOTENC	Error_train	Error_test	FP	FN
Logis	0,69 %	0,99 %	4	7
KPP	0,23 %	0,54 %	3	3
Arbre	0,23 %	1,08 %	9	3
Bagging	0,23 %	0,81 %	6	3
Boosting	0,23 %	0,54 %	3	3
Forêts n=50	0,23 %	0,45 %	2	3
SVM	0,27 %	0,36 %	1	3

(Figure 17) Erreurs d'entraînement et de test pour les différents modèles (SMOTENC).

On constate que les modèles ont été mieux entrainés avec les données obtenues avec SMOTENC qu'avec NearMiss. Excepté pour le modèle de SVM, les résultats sont à peine moins bien qu'avec la fonction **train_test_split** (figure 17). Regardons les courbes ROC :



(Figure 18) Courbes ROC des différents modèles (SMOTENC).

Pour tous les modèles, les AUC sont quasiment égaux à 1 (figure 18). Les résultats obtenus sont assez équivalents aux résultats obtenus sans rééquilibrage des données et sont donc viables. Si l'on souhaite effectuer une étude avec des classes équilibrées, utiliser SMOTENC est ici envisageable.

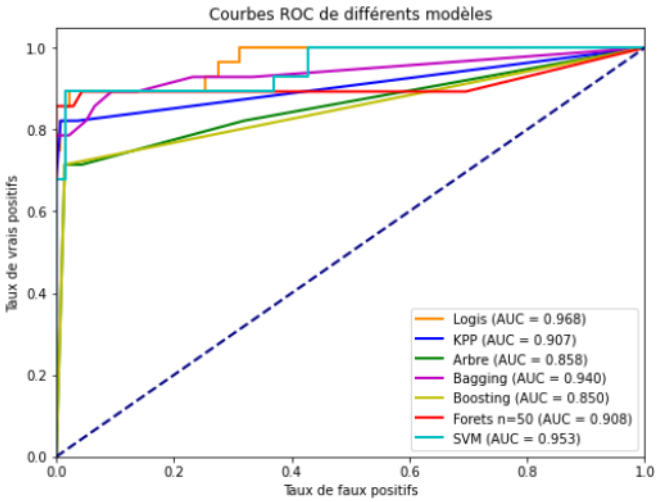
V Suppression des doublons

Nous décidons ici de voir l'impact de la suppression de ce qui semble être des doublons sur les résultats. Ainsi, nous passons d'une base de données de 4464 individus à une base de données de 662 individus. Affichons les résultats :

	Error_train	Error_test	FP	FN
Logis	2,00 %	3,60 %	1	5
KPP	3,00 %	3,60 %	1	5
Arbre	1,00 %	6,00 %	2	8
Bagging	0,20 %	4,80 %	2	6
Boosting	0,00 %	6,00 %	2	8
Forêts n=50	0,00 %	3,00 %	0	5
SVM	1,40 %	3,60 %	2	4

(Figure 19) Erreurs d'entraînement et de test pour les différents modèles (sans doublons).

On constate d'après la figure 19 que les erreurs sur les prédictions sont plus élevées après suppression des doublons qu'avant. C'est normal, la base de données contient à peu près 7 fois moins d'individus que précédemment, les modèles ont plus de mal à prédire avec un nombre réduit d'observations. Les résultats restent néanmoins très corrects.



(Figure 20) Courbes ROC des différents modèles (sans doublons).

On peut considérer qu'un modèle prédit suffisamment bien lorsque son AUC est supérieure à 0.9. En l'occurence, les modèles de forêts aléatoires, KPP, bagging, régression logistique et SVM. On notera avec la figure 20 que les 3 derniers offrent les meilleures performances en termes de prévisions (AUC de 0.94, 0.968 et 0.953).

Globalement les résultats restent très appréciables malgré le fait qu'ils restent moins bien que sans suppression des doublons.

VI Conclusion

Les modèles de régression logistique, forêts aléatoires et SVM se sont montrés être les meilleurs lors de ce projet. La régression logistique permet d'avoir facilement accès aux variables les plus influentes ce qui n'est pas le cas des deux autres modèles. Les modèles de SVM et forêts aléatoires ont obtenu les taux d'erreur sur la base de données de test les plus bas (figure 13) et globalement les AUC des 3 modèles sont presque égales à 1 (figure 14), ce qui est signe d'excellentes prédictions. Contrairement à l'arbre décisionnel simple qui permet de visualiser les variables les plus influentes (figure 9), la nature d'ensemble d'arbres de décision qui qualifie les forêts aléatoires rend l'interprétation plus compliquée au profit d'une erreur de prédicition plus faible. De plus et comme mentionné précédemment, l'ajout d'un grand nombre d'arbres peut ne pas toujours entrainer une diminution significative de l'erreur de prédiction.
Quand au modèle de SVM qui a fourni les meilleurs résultats au cours de ce projet, la nature géométrique du modèle rend l'interprétation des facteurs ou des caractéristiques qui contribuent aux prédictions de ce dernier plus complexe. De plus, les résultats peuvent varier en fonction du choix des hyperparamètres C et gamma, il faut donc optimiser ces

paramètres pour obtenir de meilleures performances, ce qui a été ici assez couteux en temps pour une base de données de 4464 individus.
Les classes étant déséquilibrées, on a décidé d'appliquer un rééquilibrage avec SMOTENC et les résultats obtenus ont été tout aussi corrects que sans rééquilibrage.
Enfin, le choix du modèle à utiliser dépend des résultats voulus. Si l'on souhaite avoir des bonnes erreurs de prédictions tout en sachant facilement quelles variables sont plus ou moins influentes sur la prédiction alors on utilisera la régression logistique. Si l'on souhaite avoir l'erreur de prédiction la plus petite possible alors mieux vaut utiliser les modèles de forêts aléatoires et SVM bien que les interprétations des facteurs qui contribuent aux prédictions des modèles soient plus complexes.