

ENSIIE ET UEVE

RAPPORT DE PROJET APPRENTISSAGE AUTOMATIQUE

Compétition Kaggle : IEEE-CIS Fraud Detection

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
testRT12.csv	a few seconds ago	1 seconds	17 seconds	0.912984
Complete				
Jump to your position on the leaderboard ▼				

Alexandre Prevot
Romain Loirs
Mathieu Aubry
Paul Bourmaud

Encadrants : M.JANODET
M.HANCZAR
Mme.BOURGEAIS

Table des matières

Introduction	4
1 Étude statistique	5
1.1 Les datasets	5
1.2 Les variables à notre disposition	5
1.2.1 Les variables du tableau transaction	5
1.2.2 Les variables du tableau identity	5
1.3 Étude de la variable cible	6
1.4 Étude des variables explicatives	6
1.4.1 Étude des variables de type numérique	6
1.4.1.1 Études des valeurs non renseignées	6
1.4.1.2 Études des histogrammes et des boxplots	7
1.4.2 Étude des variables de type chaîne de caractères	8
2 Prétraitement des données	10
2.1 Gestion des valeurs manquantes	10
2.2 Transformation des valeurs de type chaîne de caractères en valeurs de type numérique	10
2.3 Concaténation des tableaux	11
2.4 Utilisation de l'undersampling et de l'oversampling	11
2.5 Séparation du dataset Train en deux	12
2.6 Diminution de la dimension	12
3 Modèles utilisés	14
3.1 Modèle SVM (Support Vector Machine - machine à vecteur de support)	14
3.1.1 Démarche générale	14
3.1.2 Recherche des hyperparamètres	14
3.1.3 Entraînement du modèle et comparaison	14
3.2 Modèle du Random Forest (Forêt d'arbre décisionnel)	17
3.2.1 Principe	17
3.2.2 Recherche des hyperparamètres	17
3.2.3 Entraînement du modèle et comparaison	17
3.3 Comparaison des modèles	18
Conclusion et résultat sur Kaggle	19

A	Code Python pour le prétraitement des données	20
B	Code python pour la SVM	26
C	Code Python pour la Random Forest	31
D	Figure Random Forest	34

Table des figures

1.1	histogramme de la variable isFraud	6
1.2	histogrammes de différentes variables de type numérique	8
1.3	histogrammes de différentes variables de types chaîne de caractères de la table transaction	9
1.4	Histogramme de la variable P_emaildomain	9
2.1	Exemple de conversion.	10
2.2	Principe de l'oversampling et de l'undersampling	11
2.3	histogramme de la variable isFraud après l'utilisation de l'undersampling et de l'oversampling sur le dataset	12
2.4	Matrice de corrélation des variables	13
3.1	comparaison des SVM avec différents noyaux	15
3.2	comparaison des matrices de confusion pour les différents noyaux	16
3.3	Calcul du Score F1 pour les différents noyaux	16
3.4	Résultats du Random Forest	18
3.5	Résultat sur Kaggle	19
D.1	Variables les plus significatives pour le modèle	34

Introduction

Bien que peut-être encombrant (et souvent embarrassant) pour le moment, le système de prévention de la fraude permet aux consommateurs d'économiser des millions de dollars par an. Les chercheurs de l'IEEE Computational Intelligence Society (IEEE-CIS) souhaitent améliorer ce chiffre, tout en améliorant l'expérience client.

IEEE-CIS (association professionnelle de l'Institute of Electrical and Electronics Engineers) s'associe à Vesta Corporation, leader mondial des services de paiement, à la recherche des meilleures solutions pour le secteur de la prévention de la fraude.

Le but du projet est de comparer des modèles de machine learning sur un ensemble de données à grande échelle. Les données proviennent des transactions de commerce électronique réelles de Vesta et contiennent un large éventail de fonctionnalités allant du type d'appareil aux fonctionnalités du produit.

Chapitre 1

Étude statistique

1.1 Les datasets

Nous avons à notre disposition un dataset fragmenté en quatre tableaux.

- Les tableaux `train_identity` et `train_transaction` nous serviront à mettre au point un modèle
- Les tableaux `test_identity` et `test_transaction` nous serviront à tester le modèle que nous aurons trouver sur Kaggle.

Pour l'étude statistique, nous allons concaténer (verticalement) les tableaux `train_identity` et `test_identity` d'une part et `train_transaction` et `test_transaction` d'autre part. Nous appellerons respectivement, par la suite, ces deux tableaux `transaction` et `identity`. Cette concaténation nous permettra d'étudier les valeurs des variables afin, par la suite, de réaliser les mêmes modifications sur les quatre tableaux.

1.2 Les variables à notre disposition

1.2.1 Les variables du tableau `transaction`

Nous avons à notre disposition 394 variables pour un total de 590540 observations :

- `TransactionDT` : intervalle de temps à partir d'une data/heure de référence
- `TransactionAMT` : montant du paiement de la transaction en USD
- `ProductCD` : code produit, le produit pour chaque transaction
- `card1 - card6` : les informations de la carte de paiement, telles que le type de carte, la catégorie de carte, la banque d'émission, le pays, etc.
- `addr` : adresse
- `dist` : distance
- `P_ and (R_)` `emaildomain` : domaine de messagerie de l'acheteur et du destinataire
- `C1-C14` : comptage, comme le nombre d'adresses associées à la carte de paiement, etc. La signification réelle est masquée.
- `D1-D15` : `timedelta`, comme les jours entre la transaction précédente, etc.
- `M1-M9` : `match`, comme les noms sur la carte et l'adresse, etc
- `Vxxx` : Vesta a conçu des fonctionnalités riches, y compris le classement, le comptage et d'autres relations d'entité.

1.2.2 Les variables du tableau `identity`

Nous avons à notre disposition 40 variables :

- `id_01-id_38` : des identifiants relatifs à la transaction.
- `DeviceType`, `DeviceInfo` : informations relatives à la nature du terminal à partir duquel la transaction a lieu.

1.3 Étude de la variable cible

Le but du projet est de déterminer si une transaction est une fraude ou non. Ainsi, notre but est de prédire la variable `isFraud` grâce aux autres variables du dataset. Commençons par étudier cette variable. L'histogramme présenté sur la Figure 1.1 ci-dessous nous apprend que cette variable peut prendre deux valeurs : 0 ou 1.

- 0 si la transaction n'est pas une fraude
- 1 si la transaction est une fraude

L'histogramme nous apprend également que les classes sont déséquilibrées. 96% des observations appartiennent à la classe dominante. C'est à dire celle des non fraudes. Cependant, c'est la classe minoritaire qui nous intéresse. C'est à dire celle des fraudes.

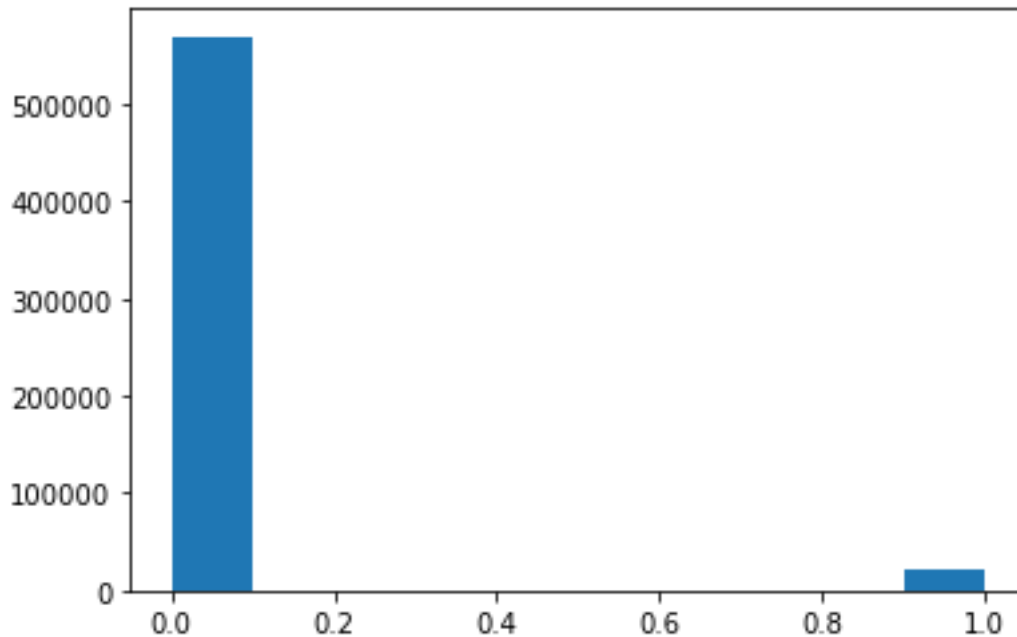


FIGURE 1.1 – histogramme de la variable `isFraud`

1.4 Étude des variables explicatives

Toutes les autres variables sont des variables explicatives potentielles sauf la variable `TransactionID` qui est un identifiant qui sera utile pour la jointure des tables `train_transaction` et `train_identity`. Une première analyse des données nous montre que les variables ne sont pas toutes du même type. Certaines variables sont de types numériques et d'autres sont des chaînes de caractères, on parle de variables discrètes.

1.4.1 Étude des variables de type numérique

1.4.1.1 Études des valeurs non renseignées

Dans un premier temps, nous avons étudié les variables qui sont des valeurs numériques. Nous les avons classées selon la proportion de valeurs manquantes.

- **Variables contenant moins de 1% de valeurs non renseignées :** `TransactionDT`, `TransactionAmt`, `card1`, `V95`, `V96`, `V97`, `V98`, `V99`, `V100`, `V101`, `V102`, `V103`, `V104`, `V105`, `V106`, `V107`, `V108`, `V109`, `V110`, `V111`, `V112`, `V113`, `V114`, `V115`, `V116`, `V117`, `V118`, `V119`, `V120`, `V121`, `V122`,

V123, V124, V125, V126, V127, V128, V129, V130, V131, V132, V133, V134, V135, V136, V137, card3, C13, D1, V281, V282, V283, V288, V289, V296, V300, V301, V313, V314, V315, card5, V279, V280, V284, V285, V286, V287, V290, V291, V292, V293, V294, V295, V297, V298, V299, V302, V303, V304, V305, V306, V307, V308, V309, V310, V311, V312, V316, V317, V318, V319, V320, V321, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C14,

- **Variables contenant moins de 10% de valeurs non renseignées :** card2, D10, V12, V13, V14, V15, V16, V17, V18, V19, V20, V21, V22, V23, V24, V25, V26, V27, V28, V29, V30, V31, V32, V33, V34, V53, V54, V55, V56, V57, V58, V59, V60, V61, V62, V63, V64, V65, V66, V67, V68, V69, V70, V71, V72, V73, V74, D15, V75, V76, V77, V78, V79, V80, V81, V82, V83, V84, V85, V86, V87, V88, V89, V90, V91, V92, V93, V94,
- **Variables contenant moins de 50% de valeurs non renseignées :** addr1, addr2, D4, V35, V36, V37, V38, V39, V40, V41, V42, V43, V44, V45, V46, V47, V48, V49, V50, V51, V52, D11, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, D3, isFraud, D2, D5
- **Variables contenant moins de 75% de valeurs non renseignées :** dist1, V220, V221, V222, V227, V234, V238, V239, V245, V250, V251, V255, V256, V259, V270, V271, V272, V167, V168, V172, V173, V176, V177, V178, V179, V181, V182, V183, V186, V187, V190, V191, V192, V193, V196, V199, V202, V203, V204, V205, V206, V207, V211, V212, V213, V214, V215, V216, V169, V170, V171, V174, V175, V180, V184, V185, V188, V189, V194, V195, V197, V198, V200, V201, V208, V209, V210 id_01, id-01, id_11, id_02, id_17, id_19, id_20, id-02, id_05, id_06, id-11, id-17, id-19, id-20, id-05, id-06, id-13, id_13, id_14, id_32, id_09, id_10, id-09, id-10,
- **Variables contenant plus de 75% de valeurs non renseignées :** V217, V218, V219, V223, V224, V225, V226, V228, V229, V230, V231, V232, V233, V235, V236, V237, V240, V241, V242, V243, V244, V246, V247, V248, V249, V252, V253, V254, V257, V258, V260, V261, V262, V263, V264, V265, V266, V267, V268, V269, V273, V274, V275, V276, V277, V278, D6, D13, D14, V322, V323, V324, V325, V326, V327, V328, V329, V330, V331, V332, V333, V334, V335, V336, V337, V338, V339, V143, V144, V145, V150, V151, V152, V159, V160, V164, V165, V166, V138, V139, V140, V141, V142, V146, V147, V148, V149, V153, V154, V155, V156, V157, V158, V161, V162, V163, D8, D9, D12, D7 dist2 id-14, id-32, id-03, id-04, id_03, id_04, id-18, id_18, id_22, id_26, id_21, id_07, id_08, id_25, id-22, id-07, id-08, id-21, id-26, id-25, id_24, id-24,

On peut remarquer que la proportion de valeurs manquantes est très inégale d’une variable à une autre. La quantité de variables ayant plus de 75% de valeurs non renseignées est très importantes.

1.4.1.2 Études des histogrammes et des boxplots

Dans un deuxième temps, nous avons réalisé un histogramme et un boxplot pour chaque variable dont une partie est présentée sur la Figure 1.2.

Ainsi Sur la figure 1.2a, nous avons représenté l’histogramme de la variable TransactionAmt (qui représente le montant de la transaction en dollars). Nous remarquons que les trois premiers quartiles sont extrêmement proches par rapport à l’étalement totale des données. En effet, les valeurs des trois premiers quartiles ne dépassent pas 1.000\$ et les valeurs extrêmes sont comprises ente 1.000 et 15.000\$ sauf un montant qui dépasse les 30.000\$.

Ensuite, les variables de type card présentent des histogrammes très similaires. Un exemple est donné sur la Figure 1.2b. Les valeurs semblent réparties de manière plus ou moins uniforme.

En outre, les variables de type C et V dont des exemples d’histogrammes sont respectivement donnés sur la Figure 1.2c et la Figure 1.2d ont une valeur dominante qui écrase toutes les autres.

Enfin, certaines variables de la table identity montrent des répartitions de valeurs plus intéressantes. Certaines variables comme id-18 ont une répartition qui semble suivre une répartition semblable à une loi normale comme montré sur la figure 1.2e. De manière analogue, certaines variables comme la variable id-02 présenté sur la figure 1.2f semblent suivre une loi de type exponentielle décroissante.

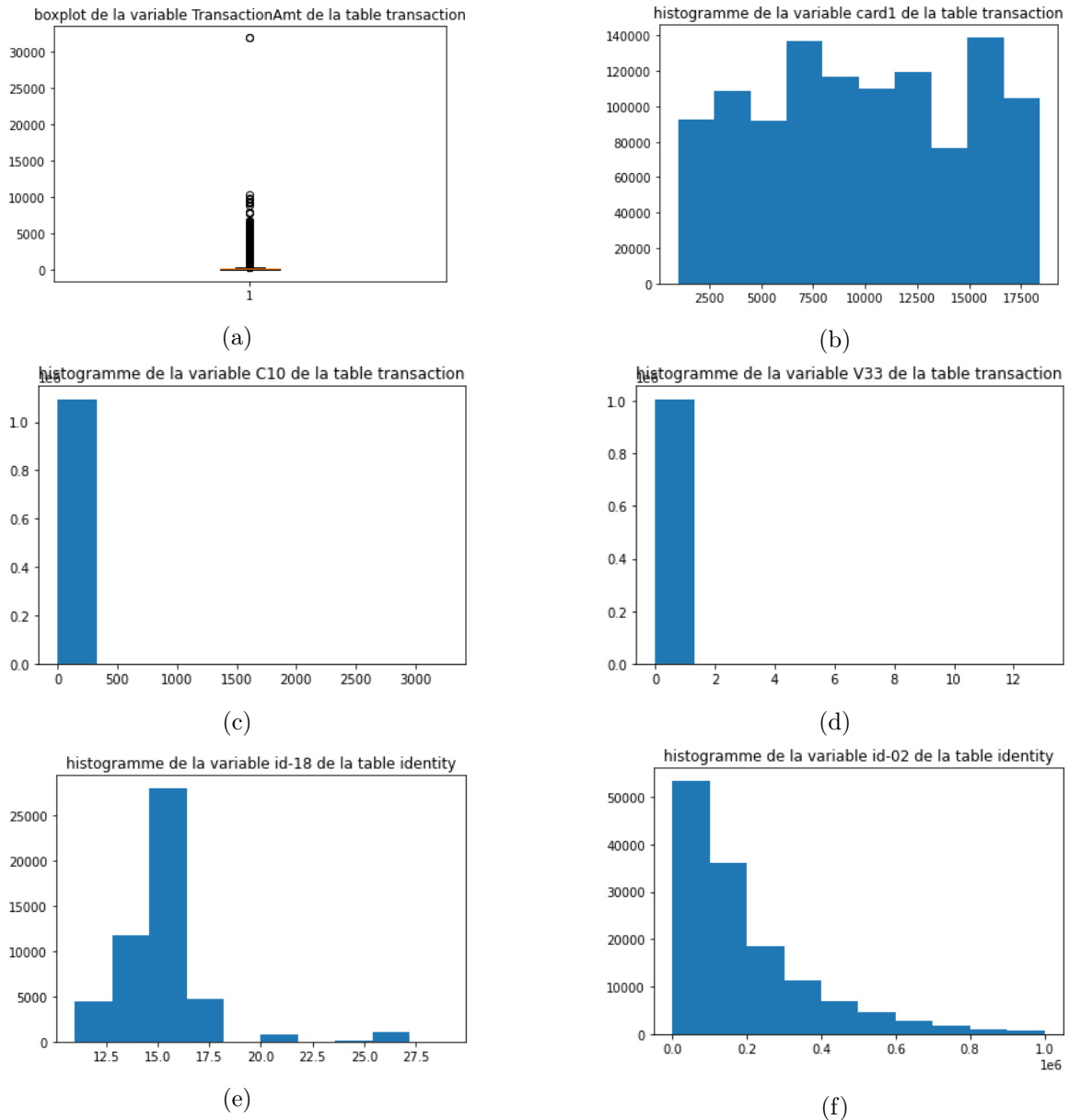


FIGURE 1.2 – histogrammes de différentes variables de type numérique

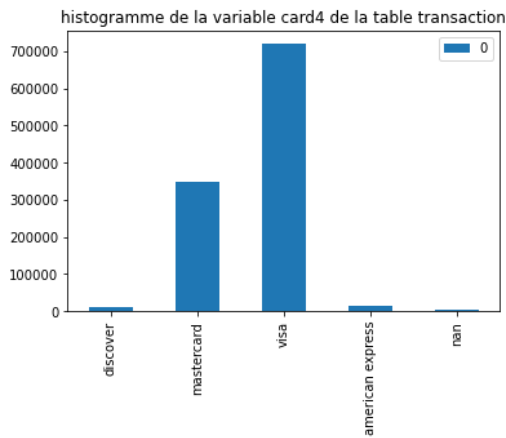
1.4.2 Étude des variables de type chaîne de caractères

Dans un premier temps, nous avons dénombré les variables qui sont de type chaîne de caractères (variables dites discrètes ou catégorielles). Celle-ci sont : ProductCD, card4, card6, P_emaildomain, R_emaildomain, M1, M2, M3, M4, M5, M6, M7, M8, M9, id_12, id_15, id_16, id_23, id_27, id_28, id_29, id_30, id_31, id_33, id_34, id_35, id_36, id_37, id_38, DeviceType, DeviceInfo, id-12, id-15, id-16, id-23, id-27, id-28, id-29, id-30, id-31, id-33, id-34, id-35, id-36, id-37, id-38

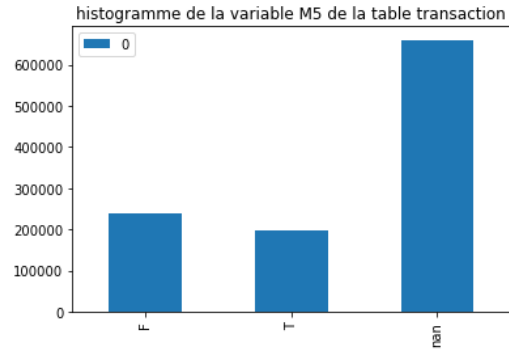
Dans un second temps, nous nous sommes intéressés, de la même manière que pour les variables de type numérique, aux histogrammes de ces variables.

D'abord, nous avons pu observer deux cas de figures pour les variables de types chaîne de caractères :

- Le premier cas est celui présenté sur la figure 1.3a. Les variables prennent un nombre fini de valeurs différentes.
- Le deuxième cas est celui présenté sur la figure 1.3b où la variable ne peut prendre que deux valeurs (sans compter les valeurs manquantes). Il s'agit donc de variables booléennes qui ont été enregistrées sous la forme d'une chaîne de caractères.



(a)



(b)

FIGURE 1.3 – histogrammes de différentes variables de types chaîne de caractères de la table transaction

Un cas particulier du premier cas que nous avons vu précédemment est le cas des variables P_emaildomain et R_emaildomain. L'étude de l'histogramme présenté sur la Figure 1.4 montre que ces variables prennent une multitude de valeurs différentes. Cependant, les domaines de messagerie comme gmail et yahoo sont dominants. On pourra également remarquer que malgré la diversité des valeurs, certaines sont très semblables tel que yahoo.com, yahoo.fr, yahoo.es, yahoo.de.

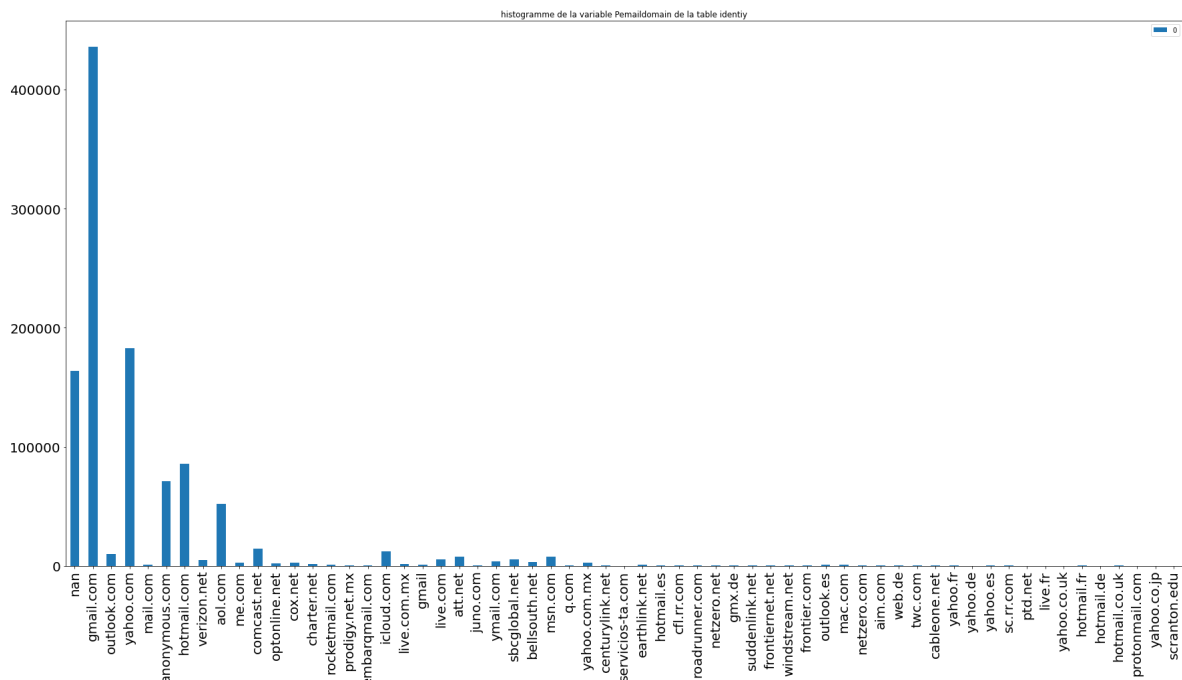


FIGURE 1.4 – Histogramme de la variable P_emaildomain

Chapitre 2

Prétraitement des données

2.1 Gestion des valeurs manquantes

Lors de l'utilisation d'un algorithme en Machine Learning, il est important d'identifier les données manquantes car ceux ci peuvent ne pas être utilisable s'il existe des valeurs manquantes. Or, précédemment, nous avons constaté qu'une grande quantité de variables comportaient des valeurs non renseignées. Nous avons ainsi décidé de classer les colonnes en 3 catégories comme montré aux lignes 109 à 125 de l'annexe A :

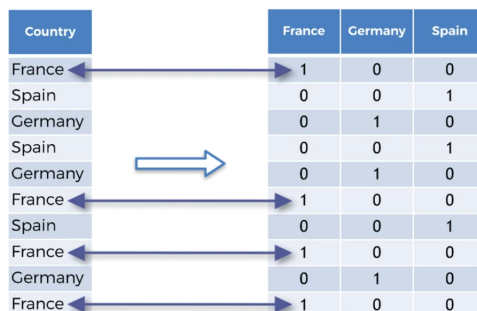
- les colonnes comportant en dessous de 15% de valeurs manquantes
- les colonnes comportant entre 15% et 60% de valeurs manquantes
- les colonnes comportant plus de 60% de valeur manquantes

Puis, nous avons développé une stratégie pour chacune de ces catégories :

- Supprimer les colonnes contenant plus de 60% de valeurs manquantes.
- Les valeurs manquantes des colonnes comportant en dessous de 15% de valeurs manquantes ont été imputées avec la moyenne
- Les valeurs manquantes des colonnes comportant entre 15% et 60% de valeurs manquantes ont été imputées avec la médiane

Nous avons réalisé les imputations des valeurs avec la médiane et la moyenne à l'aide de la fonction SimpleImputer de la librairie sklearn.impute comme montré aux lignes 137 à 161 de l'annexe A.

2.2 Transformation des valeurs de type chaîne de caractères en valeurs de type numérique



Country	France	Germany	Spain
France	1	0	0
Spain	0	0	1
Germany	0	1	0
Spain	0	0	1
Germany	0	1	0
France	1	0	0
Spain	0	0	1
France	1	0	0
Germany	0	1	0
France	1	0	0

FIGURE 2.1 – Exemple de conversion.

Afin d'utiliser un modèle, il a fallu convertir les valeurs de type chaîne de caractères en valeurs numériques. Pour cela, pour chaque variables de type string, nous l'avons remplacée par un ensemble des nouvelles variables représentant les occurrences des valeurs de l'ancienne variable. Une valeur booléenne vient indiquer la valeur de l'ancienne variable. Un exemple est donné sur la figure 2.1. Nous avons réalisé cette partie à l'aide en particulier d'une fonction de la librairie `sklearn.preprocessing` qui se nomme `LabelEncoder` (cf. ligne 162-229 annexe A).

2.3 Concaténation des tableaux

Afin de pouvoir utiliser des algorithmes de Machine Learning, nous avons concaténé les tableaux `train_identity` et `train_transaction` en réalisant une jointure externe à l'aide de la clé primaire `ID_Transaction` commune aux deux tableaux (cf. ligne 227-229 annexe A). Or, cette jointure fait apparaître des valeurs manquantes. En effet, les deux tableaux n'ont pas la même taille. Afin de remédier à ce problème, nous avons remplacé les valeurs manquantes de la colonne par la médiane de celle-ci (cf. ligne 244-246 annexe A). De la même manière, nous avons concaténé `test_identity` et `test_transaction`. Nous avons ainsi à notre disposition un tableau d'apprentissage et un tableau de test pour soumettre nos résultats sur Kaggle.

2.4 Utilisation de l'undersampling et de l'oversampling

Comme précédemment constaté, la proportion de fraudes et des non fraudes est déséquilibrée. Or, la classe qui nous intéresse réellement est la classe des non fraudes. Afin de développer un modèle qui prédit correctement la classe minoritaire et non pas uniquement la classe majoritaire, nous avons utilisé un modèle hybride d'undersampling et d'oversampling (cf. ligne 44-53 annexe C).

Comme montré sur la Figure 2.2 :

- L'undersampling consiste à éliminer aléatoirement des exemples de la classe majoritaire pour diminuer leur effet sur le modèle. En revanche, tous les exemples de la classe minoritaire sont conservés.
- L'oversampling consiste à générer des données supplémentaires (copies, données synthétiques) de la classe minoritaire pour augmenter leur effet sur le modèle. En revanche, tous les cas de la classe majoritaire sont conservés.

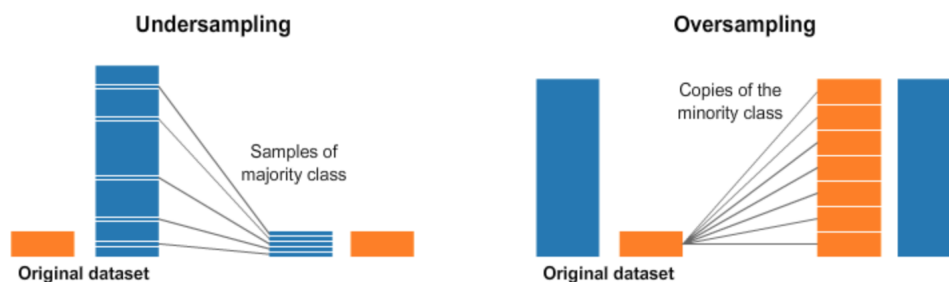


FIGURE 2.2 – Principe de l'oversampling et de l'undersampling

Le résultat de l'utilisation de ces deux méthodes sur notre dataset est montré sur l'histogramme de la Figure 2.3. Les deux classes possèdent maintenant environ 20.000 observations chacune.

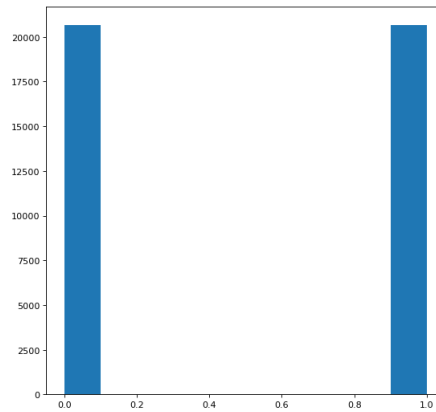


FIGURE 2.3 – histogramme de la variable isFraud après l'utilisation de l'undersampling et de l'oversampling sur le dataset

2.5 Séparation du dataset Train en deux

Nous avons séparé notre dataset train en deux :

- Ensemble d'apprentissage : sous-ensemble destiné à l'apprentissage d'un modèle.
- Ensemble d'évaluation aussi appelé ensemble de validation : sous-ensemble destiné à l'évaluation du modèle.

Chacun des dataset contient respectivement 75% et 25% des observations du dataset original. Cette séparation a été possible car notre dataset était très volumineux.

(cf. ligne 232-251 annexe A)

2.6 Diminution de la dimension

Nous avons remarqué en réalisant la matrice de corrélation que les variables sont très corrélées entre elle comme montré sur la Figure 2.4. Une multicollinéarité prononcée peut s'avérer problématique, car elle peut augmenter la variance des coefficients de régression et les rendre instables et difficiles à interpréter.

Nous avons essayé deux techniques qui n'ont pas été fructueuses pour réduire la dimension :

- technique 1 : parcourir la matrice de corrélation et pour tout les couples de variables corrélés, sélectionner une variable et la supprimer. Cette technique ne nous semblait pas correcte car très aléatoire. En effet, nous choissions toujours la première variable du couple.
- technique 2 : ACP (analyse en composante principale) qui est une méthode de la famille de l'analyse des données qui consiste à transformer des variables « corrélées » en nouvelles variables décorrélées les unes des autres. Ces nouvelles variables sont nommées les « composantes principales ».

Finalement, nous avons décidé de ne pas réduire le nombre de variable du dataset. En effet, Les deux techniques n'ont pas été utile pour avoir de meilleurs performances.

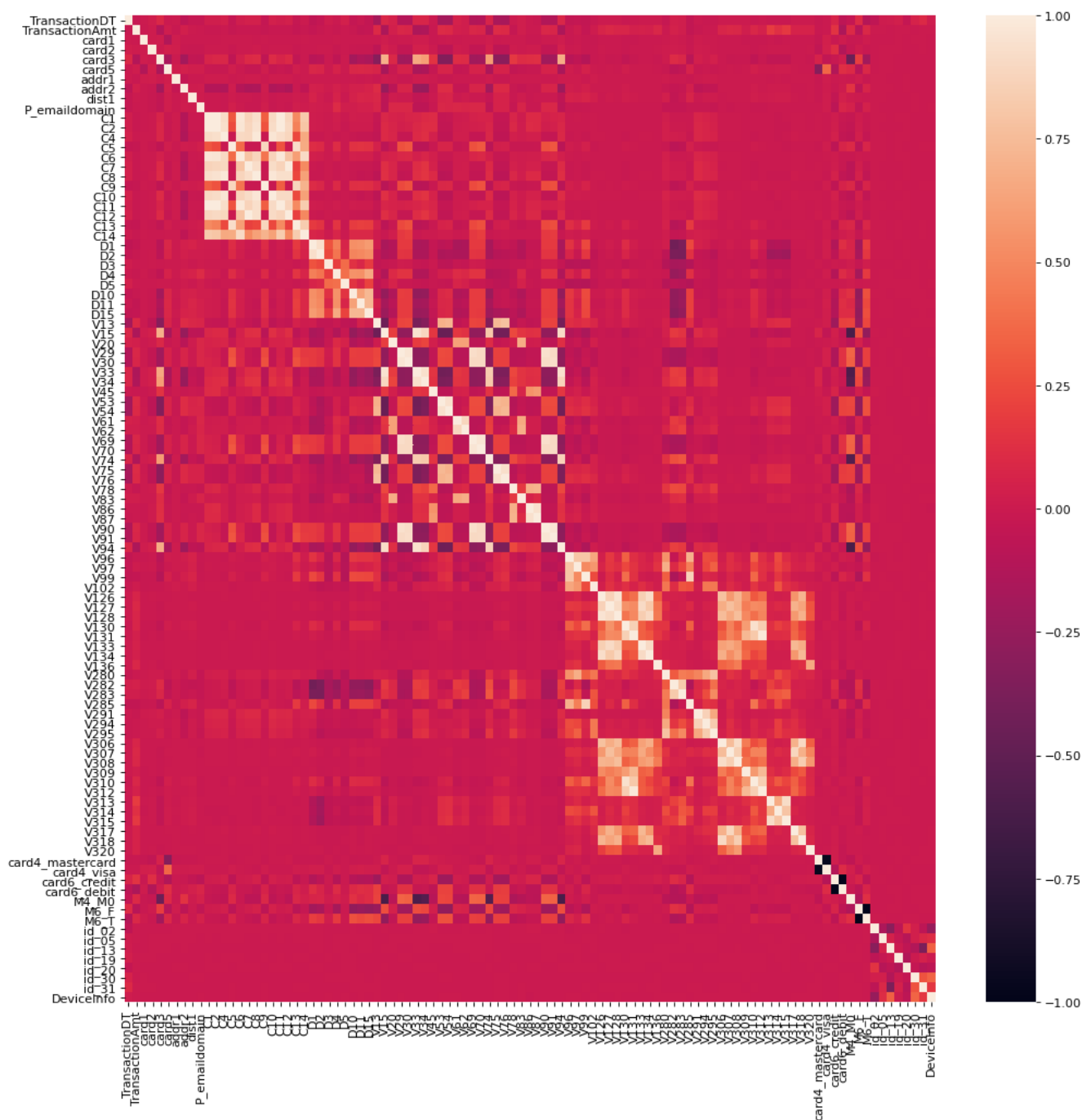


FIGURE 2.4 – Matrice de corrélation des variables

Chapitre 3

Modèles utilisés

3.1 Modèle SVM (Support Vector Machine - machine à vecteur de support)

3.1.1 Démarche générale

Nous avons réalisé une classification SVM à l'aide de quatre noyaux différents : le noyau linéaire, le noyau gaussien, le noyau puissance (noyau polynomial) et le noyau sigmoïde. Nous allons déterminer le meilleur modèle possible pour chaque noyau puis les comparer afin de choisir le meilleur modèle. Le code correspondant aux différentes étapes décrites dans les sections ultérieures est donné dans l'annexe B. Nous avons principalement utilisé la librairie sklearn de python.

3.1.2 Recherche des hyperparamètres

Pour chaque noyau, nous allons tester différentes valeurs pour la pénalisation. Les paramètres C testés seront : 0.001, 0.00464159, 0.0215443, 0.1, 0.464159, 2.15443, 10, 46.4159, 215.443, 1000. De plus, pour les noyaux gaussien et puissance nous allons respectivement tester les paramètres :

- la variance (gamma) qui pourra prendre les valeurs : 0.01, 0.0215443, 0.0464159, 0.1, 0.215443, 0.464159, 1, 2.15443, 4.64159, 10
- le degré du polynôme qui pourra prendre les valeurs : 2, 3, 5, 7, 10

Afin de départager les différents hyperparamètres, nous allons créer pour chaque noyau une grille de recherche qui trouvera le (couple) de paramètre(s) qui conviendra le mieux. Nous allons ainsi faire tourner une recherche grille avec un dataset de validation qui est extrait du dataset d'apprentissage et indépendant de ce dernier. Les paramètres retenus seront ceux dont le modèle associé aura l'AUC (area under the curve) le plus élevé. La recherche sur grille nous renvoie

- The optimal parameters for a linear kernel are 'C' : 0.00464158833612777 with a score of 0.80
- The optimal parameters for a gaussian kernel are 'C' : 10.0, 'gamma' : 0.021544346900318832 with a score of 0.78
- The optimal parameters for a puiss kernel are 'C' : 2.154434690031882, 'degree' : 2 with a score of 0.80
- The optimal parameters for a sigmoid kernel are 'C' : 0.1 with a score of 0.79

3.1.3 Entraînement du modèle et comparaison

Lorsque nous avons obtenue les différents hyperparamètres optimaux pour chaque noyaux, nous avons pu entraîner chaque modèle à l'aide du dataset d'apprentissage. Nous obtenons ainsi quatre modèles que nous pouvons comparer. Pour cela, nous allons, à l'aide du dataset d'évaluation, réaliser des prédictions que nous allons pouvoir comparer.

Dans un premier temps, nous allons comparer nos quatre modèles à l'aide de la courbe ROC. Ainsi, sur la Figure 3.1, nous avons tracé les courbes pour les quatre noyau. Nous en avons également profité pour afficher l'AUC associé à chaque méthode. Ainsi, il paraît évident que les méthodes les plus performantes sont celles développées à l'aide du noyau gaussien et du noyau puissance car ils ont un AUC de 0.89. Ce qui est supérieur à l'AUC des méthodes utilisant les noyaux linéaire et sigmoïde qui ont un AUC respectivement de 0.85 et 0.64.

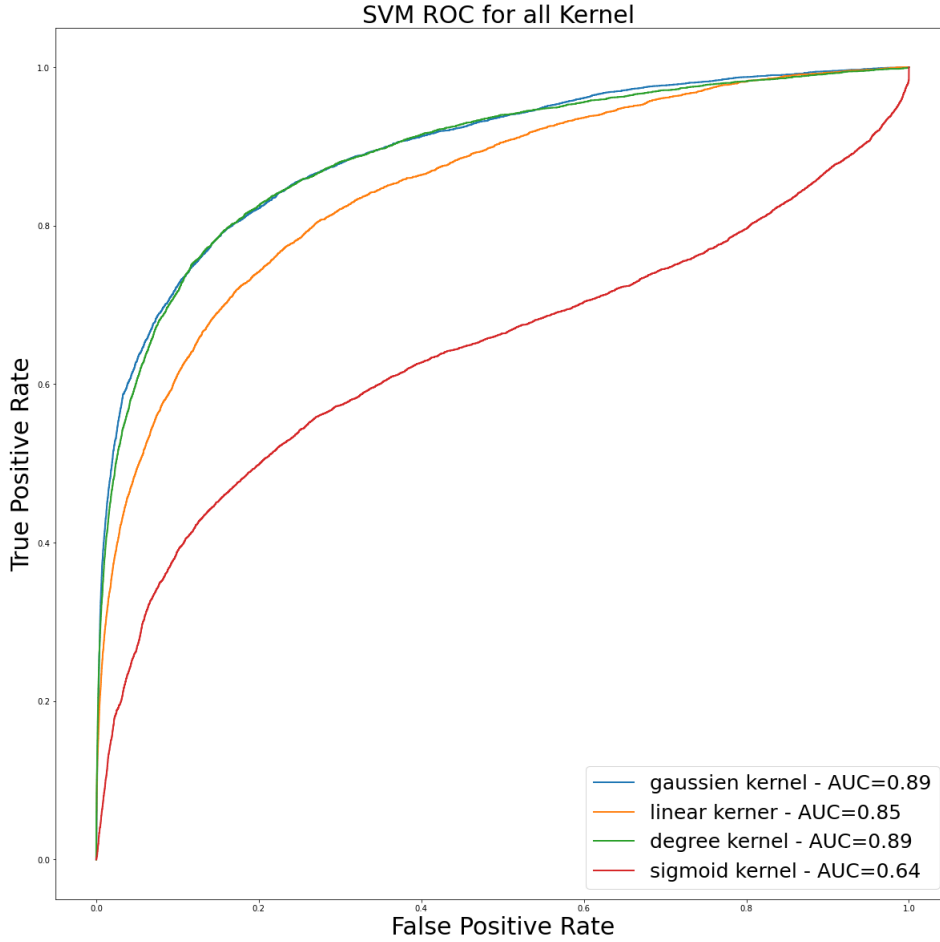


FIGURE 3.1 – comparaison des SVM avec différents noyaux

Par la suite, nous nous sommes intéressés aux matrices de confusions réalisées sur le dataset de test pour chaque noyau. Celles-ci sont données sur la Figure 3.2

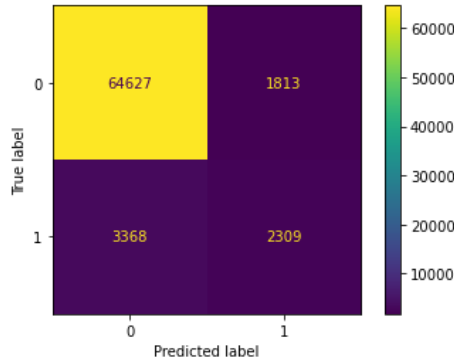
A l'aide de ces matrices, nous avons pu calculer un autre indicateur que l'AUC : le F1-Score dont la formule est rappelée par l'équation (3.1).

$$F1_score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (3.1)$$

$$Recall = \frac{TP}{TP + FN}$$

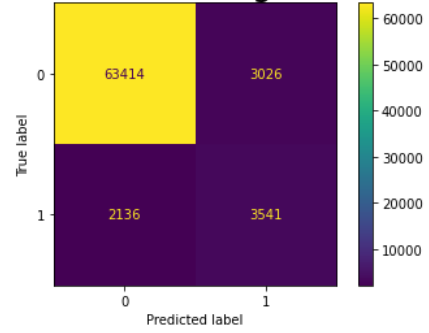
$$Precision = \frac{TP}{TP + FP}$$

confusion matrix for a linear kernel



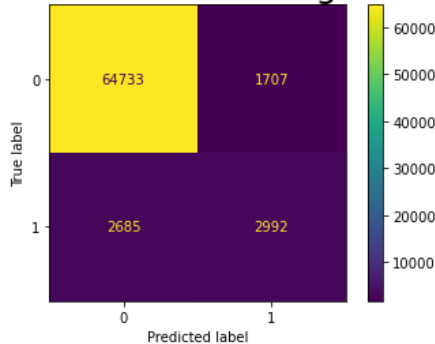
(a)

confusion matrix for a gaussian kernel



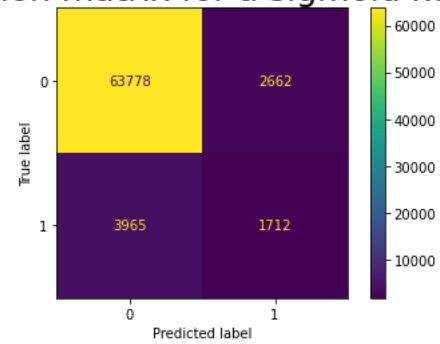
(b)

confusion matrix for a degree kernel



(c)

confusion matrix for a sigmoid kernel



(d)

FIGURE 3.2 – comparaison des matrices de confusion pour les différents noyaux

Les résultat du F1-score sont donnés sur la Figure 3.3. Cette métrique est plus indiquée que d'autres métriques plus simples tel que l'accuracy car celle-ci est souvent proche de 1 dans le cas de deux classes déséquilibrées. Nous pouvons remarquer que les F1-score nous conforte dans l'idée que les noyaux sigmoïdes et linéaires sont les moins adaptés au problème. De plus, ce F1-score nous permet de discriminer les deux derniers noyaux. En effet, le F1-score du noyau gaussien est plus élevé que celui du noyau polynomial. Nous pouvons donc légitimement préférer le modèle gaussien aux autres.

Calcul du F1-Score

Noyau	TN	FP	FN	TP	Precision	Rappel	F1-Score
Linéaire	64627	3368	1813	2309	0,406728906112383	0,560164968461912	0,471272578834575
Gaussien	63414	2136	3026	3541	0,623744935705478	0,539211207552916	0,578405749754982
Degree	64733	2685	1707	2992	0,527038929011802	0,636731219408385	0,576715497301465
Sigmoid	63778	3965	2662	1712	0,301567729434561	0,391403749428441	0,340662620634763

FIGURE 3.3 – Calcul du Score F1 pour les différents noyaux

3.2 Modèle du Random Forest (Forêt d'arbre décisionnel)

3.2.1 Principe

La Random Forest est un algorithme incontournable en machine learning. Random Forest signifie « forêt aléatoire ». Proposé par Leo Breiman en 2001, c'est un algorithme qui se base sur l'assemblage d'arbres de décision. Il est assez intuitif à comprendre, rapide à entraîner et il produit des résultats généralisables. Seul bémol, la Random Forest est une boîte noire qui donne des résultats peu lisibles, c'est-à-dire peu explicatifs.

Un Random Forest est constituée d'un ensemble d'arbres de décision indépendants. Chaque arbre dispose d'une vision parcellaire du problème du fait d'un double tirage aléatoire :

- un tirage aléatoire avec remplacement sur les observations (les lignes de la base de données). Ce processus s'appelle le tree bagging,
- un tirage aléatoire sur les variables (les colonnes de votre base de données). Ce processus s'appelle le feature sampling.

A la fin, tous ces arbres de décisions indépendants sont assemblés. La prédiction faite par la Random Forest pour des données inconnues est alors le vote (dans le cas d'un problème de classification, sinon il s'agit de la moyenne) de tous les arbres.

la Random Forest fonctionne sur ce principe : plutôt que d'avoir un estimateur complexe capable de tout faire, la Random Forest utilise plusieurs estimateurs simples (de moins bonne qualité individuelle). Chaque estimateur a une vision parcellaire du problème. Ensuite, l'ensemble de ces estimateurs est réuni pour obtenir la vision globale du problème. C'est l'assemblage de tous ces estimateurs qui rend extrêmement performante la prédiction. Nous avons utilisé la fonction `sklearn.ensemble.RandomForestClassifier`. (cf. annexe C)

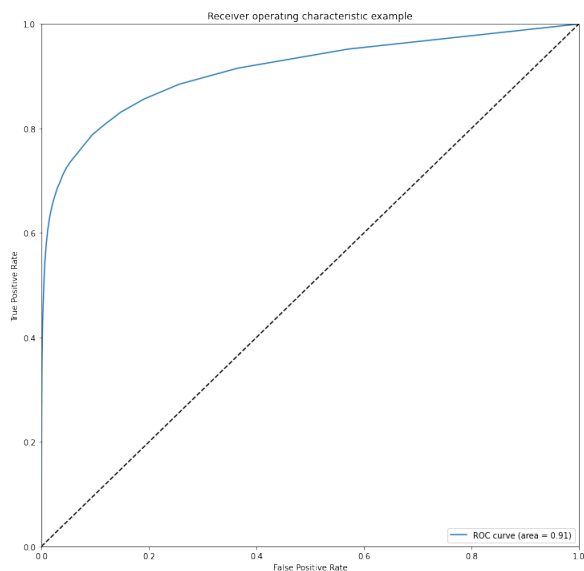
3.2.2 Recherche des hyperparamètres

Nous avons utilisé la fonction `hpsklearn` de la librairie `sklearn` pour réaliser une recherche de l'hyperparamètre (cf. ligne 71 à 90 annexe C) Les hyper-paramètres les plus importants d'une forêt aléatoire qui peuvent être réglés sont :

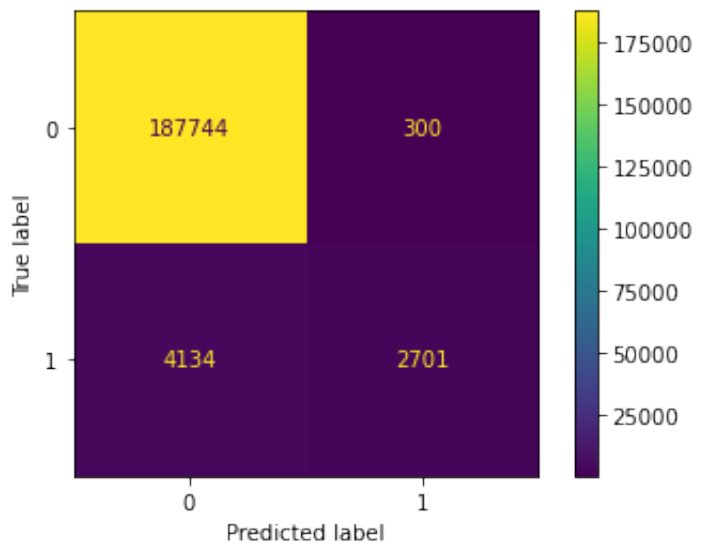
- Le nombre d'arbres de décision dans la forêt (dans Scikit-learn, ce paramètre est appelé `n_estimators`)
- La profondeur maximale des arbres individuels. Plus un arbre individuel est grand, plus il a de chances de surappliquer les données d'entraînement.
- Le minimum d'échantillons à fractionner au niveau d'un nœud interne des arbres. En jouant avec ce paramètre et le précédent, nous pourrions régulariser les arbres individuels si nécessaire.
- Nombre maximum de nœuds feuilles. Dans Random Forest, ce n'est pas si important, mais dans un arbre de décision individuel, cela peut également grandement aider à réduire le sur-ajustement et également à augmenter l'explicabilité de l'arbre en réduisant le nombre possible de chemins vers les nœuds feuilles.
- Nombre d'entités aléatoires à inclure à chaque nœud pour le fractionnement.
- La taille de l'ensemble de données bootstrap avec lequel entraîner chaque arbre de décision.
- Les critères de fractionnement sur chaque nœud (Gini ou Entropy pour une tâche de classification, ou le MSE ou MAE pour la régression)

3.2.3 Entraînement du modèle et comparaison

Nous avons par la suite utilisé la même méthodologie que pour le modèle SVM. Dans un premier temps, nous avons tracé la courbe ROC et la matrice de confusion présentées respectivement sur les figure 3.4a et 3.4b



(a) Courbe ROC pour la Random Forest



(b) Matrice de confusion pour la Random Forest

FIGURE 3.4 – Résultats du Random Forest

Ainsi, nous avons calculer les mêmes métriques que pour la SVM. C'est à dire :

- l'AUC : 0.91
- le F1_score= 0.5492069947132981

La Random Forest nous a également permis de déterminer quelle sont les variables qui sont les plus significatives pour déterminer si une transaction est une fraude ou non. Les 20 variables les plus significatives sont données sur l'annexe D.

3.3 Comparaison des modèles

Afin de faire un choix entre le modèle SVM avec noyau gaussien et la Random Forest. Nous allons comparer des indicateurs. Nous pouvons remarquer dans un premier temps que le modèle développé avec la Random Forest a un meilleur AUC mais un F1_score moins bon.

	SVM	Random Forest
F1_score	0.58	0.55
AUC	0.89	0.91

Comme nous avons deux modèles assez proche, nous nous en sommes remis aux travaux déjà réalisés par d'autre groupes sur Kaggle qui ont plébiscités la Random Forest. Ainsi, nous avons décidé d'opter pour la Random Forest.

Conclusion et résultat sur Kaggle

Le contexte de la compétition nous a plongé dans un environnement aussi rude que passionnant où l'on a été amenés à concrétiser et approfondir des domaines que nous n'avions exploré que d'un point de vue théorique jusqu'à ce jour. Le fait de pouvoir comparer notre travail avec ceux de plusieurs centaines de professionnels du domaine fut très stimulant. En plus d'orienter certaines de nos méthodes cela nous a donné un aperçu de ce qu'il était possible d'accomplir pour des personnes qui travaillent dans le domaine du data science au quotidien.

Ce projet étant complet, dans le sens où nous partions de rien et avions pour objectif de terminer avec des résultats, nous avons dû passer par toutes les étapes nécessaires au bon déroulement de celui-ci. Nous avons dû réaliser l'analyse du problème et des données, se documenter sur les méthodes à employer, appliquer ces méthodes et finalement traiter les résultats. Nous nous sommes donc organisés pour répartir toutes ces étapes sur toute la durée du projet.

Nos résultats ne nous ont, certes, pas permis d'atteindre le haut du classement sur Kaggle mais nous avons appris à exploiter des méthodes de machine Learning comme la SVM et la Random Forest.

Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
testRT12.csv	a few seconds ago	1 seconds	17 seconds	0.912984
Complete				
Jump to your position on the leaderboard ▼				

FIGURE 3.5 – Résultat sur Kaggle

Annexe A

Code Python pour le prétraitement des données

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jan 19 13:27:54 2021
4
5  @author: maubr
6  """
7  import numpy as np
8  import matplotlib.pyplot as plt
9  import pandas as pd
10
11 # Importing the dataset
12 train_transaction = pd.read_csv('train_transaction.csv')
13 test_transaction = pd.read_csv('test_transaction.csv')
14 train_identity = pd.read_csv('train_identity.csv')
15 test_identity = pd.read_csv('test_identity.csv')
16
17 #on drop les TransactionId car elle sont inutile pour la prediction
18 train_transaction = train_transaction.drop(columns='TransactionID')
19 test_transaction = test_transaction.drop(columns='TransactionID')
20 train_identity = train_identity.drop(columns='TransactionID')
21 test_identity = test_identity.drop(columns='TransactionID')
22
23 #ce que l'on veut predire notre target
24 y_train = train_transaction["isFraud"]
25
26 #on l'enleve de notre model d'entrainement sinon on aurais un model
   overfit
27 train_transaction = train_transaction.drop(columns = ["isFraud"])
28
29 #on met tout les nom des collones au m me format
30 test_identity.columns = ['id_01', 'id_02', 'id_03', 'id_04', 'id_05', 'id_06', 'id_07', 'id_08',
31                          'id_09', 'id_10', 'id_11', 'id_12', 'id_13', 'id_14', 'id_15', 'id_16',
32                          'id_17', 'id_18', 'id_19', 'id_20', 'id_21', 'id_22', 'id_23', 'id_24',
```

```

33         'id_25', 'id_26', 'id_27', 'id_28', 'id_29', 'id_30', 'id_31', '
        id_32',
34         'id_33', 'id_34', 'id_35', 'id_36', 'id_37', 'id_38', '
        DeviceType',
35         'DeviceInfo']
36
37 #on concat les fichier test et train pour permettre de faire les
        operation sur 2 fichiers au lieux de 4
38 transaction_data = pd.concat([train_transaction, test_transaction])
39 identity_data = pd.concat([train_identity, test_identity])
40
41
42 del train_identity
43 del test_identity
44 del test_transaction
45 del train_transaction
46
47 #on met dans un liste les colonnes categorique dans cat_id_cols et
        numerical dans num_id_cols
48 c = (identity_data.dtypes == 'object')
49 n = (identity_data.dtypes != 'object')
50 cat_id_cols = list(c[c].index)
51 num_id_cols = list(n[n].index)
52
53 #pareil pour transaction
54 c = (transaction_data.dtypes == 'object')
55 n = (transaction_data.dtypes != 'object')
56 cat_trans_cols = list(c[c].index)
57 num_trans_cols = list(n[n].index)
58
59
60 #on regarde combien de valeur manquante categorique il y a dans le
        fichier
61 #identity
62 low_missing_cat_id_cols = []          # en dessous de 15% de valeur
        manquante
63 medium_missing_cat_id_cols = []      # entre 15% et 60% de valeur
        manquante
64 many_missing_cat_id_cols = []        # plus de 60% de valeur manquante
65
66 for i in cat_id_cols:
67     percentage = identity_data[i].isnull().sum() * 100 / len(
        identity_data[i])
68     if percentage < 15:
69         low_missing_cat_id_cols.append(i)
70     elif percentage >= 15 and percentage < 75:
71         medium_missing_cat_id_cols.append(i)
72     else:
73         many_missing_cat_id_cols.append(i)
74
75 #pour les valeurs numeriques
76 low_missing_num_id_cols = []          # en dessous de 15% de valeur
        manquante

```

```

77 medium_missing_num_id_cols = []      # entre 15% et 60% de valeur
    manquante
78 many_missing_num_id_cols = []        # plus de 60% de valeur manquante
79
80 for i in num_id_cols:
81     percentage = identity_data[i].isnull().sum() * 100 / len(
        identity_data[i])
82     if percentage < 15:
83         low_missing_num_id_cols.append(i)
84     elif percentage >= 15 and percentage < 75:
85         medium_missing_num_id_cols.append(i)
86     else:
87         many_missing_num_id_cols.append(i)
88
89
90 #la m me chose pour transaction
91 low_missing_num_trans_cols = []       # en dessous de 15% de valeur
    manquante
92 medium_missing_num_trans_cols = []    # entre 15% et 60% de valeur
    manquante
93 many_missing_num_trans_cols = []      # more than 60% missing
94
95 for i in num_trans_cols:
96     percentage = transaction_data[i].isnull().sum() * 100 / len(
        transaction_data[i])
97     if percentage < 15:
98         low_missing_num_trans_cols.append(i)
99     elif percentage >= 15 and percentage < 75:
100         medium_missing_num_trans_cols.append(i)
101     else:
102         many_missing_num_trans_cols.append(i)
103
104 print("num_trans_cols:␣\n\n")
105 print("number␣low␣missing:␣", len(low_missing_num_trans_cols), "\n")
106 print("number␣medium␣missing:␣", len(medium_missing_num_trans_cols), "\n")
107 print("number␣many␣missing:␣", len(many_missing_num_trans_cols), "\n")
108
109 low_missing_cat_trans_cols = []       # en dessous de 15% de valeur
    manquante
110 medium_missing_cat_trans_cols = []    # entre 15% et 60% de valeur
    manquante
111 many_missing_cat_trans_cols = []      # plus de 60% de valeur manquante
112
113 for i in cat_trans_cols:
114     percentage = transaction_data[i].isnull().sum() * 100 / len(
        transaction_data[i])
115     if percentage < 15:
116         low_missing_cat_trans_cols.append(i)
117     elif percentage >= 15 and percentage < 75:
118         medium_missing_cat_trans_cols.append(i)
119     else:
120         many_missing_cat_trans_cols.append(i)

```

```

121
122 #on supprime toute les colonnes qui on plus de 60% de valeur manquante
    pour les valeurs numeriques
123 transaction_data = transaction_data.drop(columns =
    many_missing_num_trans_cols)
124
125 identity_data = identity_data.drop(columns = many_missing_num_id_cols)
126
127
128
129 #comme on vient de supprimer des colonnes on refais le liste des
    colonnes numeriques du fichier
130 n = (transaction_data.dtypes != 'object')
131 num_trans_cols = list(n[n].index)
132
133 #pareil pour identity
134 n = (identity_data.dtypes != 'object')
135 num_id_cols = list(n[n].index)
136
137 #on fait la moyenne sur les colonnes qui ont moins de 15% de valeurs
    manquantes
138 from sklearn.impute import SimpleImputer
139
140 my_imputer = SimpleImputer(strategy = 'mean')
141 my_imputer.fit(transaction_data[low_missing_num_trans_cols])
142 transaction_data[low_missing_num_trans_cols] = my_imputer.transform(
    transaction_data[low_missing_num_trans_cols])
143
144
145 #pareil pour identity
146 my_imputer = SimpleImputer(strategy = 'mean')
147 my_imputer.fit(identity_data[low_missing_num_id_cols])
148 identity_data[low_missing_num_id_cols] = my_imputer.transform(
    identity_data[low_missing_num_id_cols])
149
150
151 #on fait la median sur les colonnes qui ont entre 15% et 60% de valeurs
    manquantes
152 my_imputer = SimpleImputer(strategy = 'median')
153 my_imputer.fit(transaction_data[medium_missing_num_trans_cols])
154 transaction_data[medium_missing_num_trans_cols] = my_imputer.transform(
    transaction_data[medium_missing_num_trans_cols])
155
156
157 #la m me chose pour identity
158 my_imputer = SimpleImputer(strategy = 'median')
159 my_imputer.fit(identity_data[medium_missing_num_id_cols])
160 identity_data[medium_missing_num_id_cols] = my_imputer.transform(
    identity_data[medium_missing_num_id_cols])
161
162 #CAT DATA
163 #on supprime toute les colonnes qui on plus de 60% de valeur manquante
    pour les valeurs catgoriques

```



```

164 transaction_data = transaction_data.drop(columns =
      many_missing_cat_trans_cols)
165 identity_data = identity_data.drop(columns = many_missing_cat_id_cols)
166
167 #comme on vient de supprimer des colonnes on refais le liste des
      colonnes categoriques du fichier
168 c = (transaction_data.dtypes == 'object')
169 cat_trans_cols = list(c[c].index)
170
171 c = (identity_data.dtypes == 'object')
172 cat_id_cols = list(c[c].index)
173
174 #permet de savoir quelle sont les valeurs uniques dans chaque colonne
175 for col in cat_id_cols:
176     print(col, identity_data[col].nunique(), "\n")
177
178 #On cree une liste des valeur qui un petite cardinalite pour
      transaction
179 low_card_trans_cols = ["ProductCD", "card4", "card6", "M1", "M2", "M3",
      "M4", "M6", "M7", "M8", "M9"]
180 #On cree un liste des veleur qui on un forte cardinalite pour
      transaction
181 high_card_trans_cols = ["P_emaildomain"]
182
183
184
185 for i in cat_trans_cols:
186     most_frequent_value = transaction_data[i].mode()[0]
187     print("Pour la colonne:", i, "le valeur la plus frequente est:",
      most_frequent_value, "\n")
188     transaction_data[i].fillna(most_frequent_value, inplace = True)
189
190 #on encode les colonnes avec une grosse cardinalite (
      high_card_trans_cols) avec des 0 et des 1
191 from sklearn.preprocessing import LabelEncoder
192
193 label_encoder = LabelEncoder()
194 transaction_data[high_card_trans_cols] = label_encoder.fit_transform(
      transaction_data[high_card_trans_cols])
195
196
197 #On fait la meme chose pour identity
198 for col in cat_id_cols:
199     print(col, identity_data[col].nunique(), "\n")
200
201 low_card_id_cols = ["id_12", "id_15", "id_16", "id_28", "id_29", "
      id_34", "id_35", "id_36", "id_37", "id_38", "DeviceType"]
202 high_card_id_cols = ["id_30", "id_31", "id_33", "DeviceInfo"]
203
204
205 for i in cat_id_cols:
206     most_frequent_value = identity_data[i].mode()[0]

```

```

207     print("Pour la colonne:", i, "la valeur la plus frequente est:",
           most_frequent_value, "\n")
208     identity_data[i].fillna(most_frequent_value, inplace = True)
209
210 label_encoder = LabelEncoder()
211
212 for col in high_card_id_cols:
213     identity_data[col] = label_encoder.fit_transform(identity_data[col]
214     ])
214
215
216 #On fais le Onehotencoding pour transaction
217 low_card_trans_encoded = pd.get_dummies(transaction_data[
218     low_card_trans_cols], dummy_na = False)
219 transaction_data.drop(columns = low_card_trans_cols, inplace = True)
220
221 #On fais le Onehotencoding pour identity
222 low_card_id_encoded = pd.get_dummies(identity_data[low_card_id_cols],
223     dummy_na = False)
224 identity_data.drop(columns = low_card_id_cols, inplace = True)
225
226
227 #on concat les colonnes que l'on veut de faire avec le onehotencoding
aux fichiers transaction et identity
228 transaction_concatated = pd.concat([transaction_data,
229     low_card_trans_encoded], axis = 1)
230 identity_concatated = pd.concat([identity_data, low_card_id_encoded],
231     axis = 1)
232
233
234 #on split transaction en test et en train
235 train_transaction = transaction_concatated.iloc[0:590540]
236 test_transaction = transaction_concatated.iloc[590540:]
237
238 train_identity = identity_concatated.iloc[0:144233]
239 test_identity = identity_concatated.iloc[144233:]
240
241 #puis on cree le test data et le train data
242 train_data = pd.concat([train_transaction, train_identity], axis = 1)
243
244 test_data = pd.concat([test_transaction, test_identity], axis = 1)
245
246 #on remplace les NaN par la median
247 test_data.fillna(test_data.median(), inplace=True)
248 train_data.fillna(train_data.median(), inplace=True)
249
250 test_data.to_csv(r'test_data75.csv', index = False)
251 train_data.to_csv(r'train_data75.csv', index = False)
252 y_train.to_csv(r'y_train75.csv', index = False)

```

Annexe B

Code python pour la SVM

```
1  #librairies
2  import pandas as pd
3  import numpy as np
4  from matplotlib import pyplot as plt
5  from sklearn import model_selection
6  from sklearn import svm
7  from sklearn import metrics
8  from sklearn.metrics import plot_confusion_matrix
9
10 #importation des dataset
11 datast_test = pd.read_csv('dataset_test1.csv')
12 datast_train = pd.read_csv('dataset_train1.csv')
13 datast_valid = pd.read_csv('dataset_valid1.csv')
14
15 y_train = datast_train["isFraud"]
16 y_test = datast_test["isFraud"]
17 y_valid = datast_valid["isFraud"]
18
19 del datast_train['isFraud']
20 del datast_test['isFraud']
21 del datast_valid['isFraud']
22
23 # Communun a tout les noyaux
24 score = 'roc_auc'
25 C_range = np.logspace(-3, 3, 10)
26
27 ## noyau gaussien
28
29 #recherche des hyperparametres
30 gamma_range = np.logspace(-2, 1, 10)
31 param_grid_gauss = {'C': C_range, 'gamma': gamma_range}
32 grid_gauss = model_selection.GridSearchCV(svm.SVC(kernel='rbf'),
33                                           param_grid_gauss,
34                                           cv=5,
35                                           scoring=score)
36                                           grid_gauss.fit(datast_valid[:500],
37                                                         y_valid[:500])
37 print("The optimal parameters for a gaussian kernel are {} with a score
    of {:.2f}".format(grid_gauss.best_params_, grid_gauss.best_score_))
```

```

38
39 #train du modele
40 classifieur_gauss = svm.SVC(kernel='rbf', C=grid_gauss.best_params_['C',
    ], gamma=grid_gauss.best_params_['gamma'])
41 classifieur_gauss.fit(datast_test, y_test)
42
43 # prediction
44 y_test_pred_gauss = classifieur_gauss.decision_function(datast_train)
45
46 # construire la courbe ROC du modele optimise
47 fpr_cv_gauss, tpr_cv_gauss, thr_cv_gauss = metrics.roc_curve(y_train,
    y_test_pred_gauss)
48 auc_cv_gauss = metrics.auc(fpr_cv_gauss, tpr_cv_gauss)
49 fig = plt.figure(figsize=(20, 20))
50 plt.plot(fpr_cv_gauss, tpr_cv_gauss, '-', lw=2, label='AUC=%.2f' % \
51         ( auc_cv_gauss))
52 plt.xlabel('False Positive Rate', fontsize=30)
53 plt.ylabel('True Positive Rate', fontsize=30)
54 plt.title('SVM ROC Curve for a gaussian kernel', fontsize=30)
55 plt.legend(loc="lower right", fontsize=25)
56 plt.show()
57
58 # matrice de confusion
59 fig = plt.figure(figsize=(20, 20))
60 plot_confusion_matrix(classifieur_gauss, datast_test, y_test)
61 plt.title('confusion matrix for a gaussian kernel', fontsize=25)
62 plt.show()
63
64
65 ## noyau lineaire
66
67 #recherche des hyperparametres
68 param_grid_lin = {'C': C_range}
69 grid_lin = model_selection.GridSearchCV(svm.SVC(kernel='linear'),
70                                         param_grid_lin,
71                                         cv=5,
72                                         scoring=score)
73                                         grid_lin.fit(datast_valid, y_valid)
74 print("The optimal parameters for a linear kernel are {} with a score
    of {:.2f}".format(grid_lin.best_params_, grid_lin.best_score_))
75
76
77 #train du modele
78 classifieur_lin = svm.LinearSVC(C=grid_lin.best_params_['C'], max_iter
    =1000000)
79 classifieur_lin.fit(datast_train, y_train)
80
81 #prediction
82 y_test_pred_lin = classifieur_lin.decision_function(datast_test)
83
84 # construire la courbe ROC du mod le optimise
85 fpr_cv_lin, tpr_cv_lin, thr_cv_lin = metrics.roc_curve(y_test,
    y_test_pred_lin)

```

```

86 auc_cv_lin = metrics.auc(fpr_cv_lin, tpr_cv_lin)
87 fig = plt.figure(figsize=(20, 20))
88 plt.plot(fpr_cv_lin, tpr_cv_lin, '-', lw=2, label='AUC=%.2f' % \
89         ( auc_cv_lin))
90 plt.xlabel('False Positive Rate', fontsize=30)
91 plt.ylabel('True Positive Rate', fontsize=30)
92 plt.title('SVM ROC Curve for a linear kernel', fontsize=30)
93 plt.legend(loc="lower right", fontsize=25)
94 plt.show()
95
96 # matrice de confusion
97 fig = plt.figure(figsize=(20, 20))
98 plot_confusion_matrix(classifier_lin, datast_test, y_test)
99 plt.title('confusion matrix for a linear kernel', fontsize=25)
100 plt.show()
101
102
103 ## noyau puissance
104
105 # recherche des hyperparametres
106 degree_range = np.array([2,3,5,7,10])
107 param_grid_degree = {'C': C_range, 'degree': degree_range}
108 grid_degree = model_selection.GridSearchCV(svm.SVC(kernel='rbf'),
109                                           param_grid_degree,
110                                           cv=5,
111                                           scoring=score)
112 grid_degree.fit(datast_valid, y_valid)
113 print("The optimal parameters for a puissance kernel are {} with a score of
114       {:.2f}".format(grid_degree.best_params_, grid_degree.best_score_))
115
116 # train du modele
117 classifier_degree = svm.SVC(kernel='poly', C=grid_degree.best_params_['
118       C'], degree=grid_degree.best_params_['degree'])
119 classifier_degree.fit(datast_train, y_train)
120
121 # prediction
122 y_test_pred_degree = classifier_degree.decision_function(datast_test)
123
124 # construire la courbe ROC du modele optimise
125 fpr_cv_degree, tpr_cv_degree, thr_cv_degree = metrics.roc_curve(y_test,
126       y_test_pred_degree)
127 auc_cv_degree = metrics.auc(fpr_cv_degree, tpr_cv_degree)
128 fig = plt.figure(figsize=(20, 20))
129 plt.plot(fpr_cv_degree, tpr_cv_degree, '-', lw=2, label='AUC=%.2f' % \
130         ( auc_cv_degree))
131 plt.xlabel('False Positive Rate', fontsize=30)
132 plt.ylabel('True Positive Rate', fontsize=30)
133 plt.title('SVM ROC Curve for a degree kernel', fontsize=30)
134 plt.legend(loc="lower right", fontsize=25)
135 plt.show()
136 fig = plt.figure(figsize=(20, 20))
137 plot_confusion_matrix(classifier_degree, datast_test, y_test)
138 plt.title('confusion matrix for a degree kernel', fontsize=25)

```

```

136 plt.show()
137
138
139 ## noyau sigmoide
140
141 #Recherche des hyperparametres
142 param_grid_sig = {'C': C_range}
143 grid_sig = model_selection.GridSearchCV(svm.SVC(kernel='sigmoid'),
144                                         param_grid_sig,
145                                         cv=5,
146                                         scoring=score)
147 grid_sig.fit(datast_valid, y_valid)
148 print("The optimal parameters for a sigmoid kernel are {} with a score of {:.2f}".format(grid_sig.best_params_, grid_sig.best_score_))
149
150 # Train du modele
151 classifieur_sig = svm.SVC(kernel='sigmoid', C=grid_sig.best_params_['C'])
152 classifieur_sig.fit(datast_train, y_train)
153
154 #realiser des predictions avec datast_test
155 y_test_pred_sig = classifieur_sig.decision_function(datast_test)
156
157 # construire la courbe ROC du modele optimise
158 fpr_cv_sig, tpr_cv_sig, thr_cv_sig = metrics.roc_curve(y_test,
159                                                         y_test_pred_sig)
159 auc_cv_sig = metrics.auc(fpr_cv_sig, tpr_cv_sig)
160 fig = plt.figure(figsize=(20, 20))
161 plt.plot(fpr_cv_sig, tpr_cv_sig, '-', lw=2, label='AUC=%.2f' % \
162          ( auc_cv_sig))
163 plt.xlabel('False Positive Rate', fontsize=30)
164 plt.ylabel('True Positive Rate', fontsize=30)
165 plt.title('SVM ROC Curve for a sigmoid kernel', fontsize=30)
166 plt.legend(loc="lower right", fontsize=25)
167 plt.show()
168
169 # Matrice de confusion
170 fig = plt.figure(figsize=(20, 20))
171 plot_confusion_matrix(classifieur_sig, datast_test, y_test)
172 plt.title('confusion matrix for a sigmoid kernel', fontsize=25)
173 plt.show()
174
175
176
177 # comparaison des 4 mod les
178
179 fig = plt.figure(figsize=(20, 20))
180 plt.plot(fpr_cv_gauss, tpr_cv_gauss, '-', lw=2, label='gaussien kernel \
181          - AUC=%.2f' % \
182          ( auc_cv_gauss))
183 plt.plot(fpr_cv_lin, tpr_cv_lin, '-', lw=2, label='linear kerner - AUC \
184          =%.2f' % \
185          ( auc_cv_lin))

```

```

184 plt.plot(fpr_cv_degree, tpr_cv_degree, '-', lw=2, label='degree_kernel_
    -_AUC=%.2f' % \
185             ( auc_cv_degree))
186 plt.plot(fpr_cv_sigm, tpr_cv_sigm, '-', lw=2, label='sigmoid_kernel_
    AUC=%.2f' % \
187             ( auc_cv_sigm))
188 plt.xlabel('False_Positive_Rate', fontsize=30)
189 plt.ylabel('True_Positive_Rate', fontsize=30)
190 plt.title('SVM_ROC_for_all_Kernel_', fontsize=30)
191 plt.legend(loc="lower_right", fontsize=25)
192 plt.show()

```

Annexe C

Code Python pour la Random Forest

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Thu Feb 18 21:39:54 2021
5
6  @author: romainloirs
7  """
8
9  import pandas as pd
10 import numpy as np
11 from matplotlib import pyplot as plt
12 from sklearn.model_selection import GridSearchCV
13 from sklearn.ensemble import RandomForestClassifier
14 from sklearn import metrics
15 from sklearn.metrics import plot_confusion_matrix
16 from sklearn.model_selection import train_test_split
17
18 test_data = pd.read_csv('test_data.csv')
19 train_data = pd.read_csv('train_data.csv')
20 y_train1 = pd.read_csv('y_train.csv')['isFraud']
21
22
23 correlation_matrix_test = test_data.corr()
24 correlated_features_test = set()
25 for i in range(len(correlation_matrix_test.columns)):
26     for j in range(i):
27         if abs(correlation_matrix_test.iloc[i, j]) > 0.7:
28             colname = correlation_matrix_test.columns[i]
29             correlated_features_test.add(colname)
30
31
32 train_data.drop(labels=correlated_features_test, axis=1, inplace=True)
33
34 test_data.drop(labels=correlated_features_test, axis=1, inplace=True)
35
36 #on remplace tou les NaN par la median
37 test_data.fillna(test_data.median(), inplace=True)
38 train_data.fillna(train_data.median(), inplace=True)
39
```



```

40
41 #undersampling and oversampling
42 from imblearn.under_sampling import RandomUnderSampler
43 from imblearn.over_sampling import SMOTE
44 from imblearn.pipeline import Pipeline
45
46 undersample = RandomUnderSampler()
47 oversample = SMOTE()
48
49 steps = [("u",undersample),("o", oversample)]
50 pipeline = Pipeline(steps = steps)
51 train_data, y_train1 = pipeline.fit_resample(train_data,y_train1)
52
53 #RECHERCHE DES HYPER PARAMETRE 1 AVEC GridSearchCV
54
55 param_grid = {
56     'bootstrap': [True],
57     'max_depth': [20, 23, 25, 28],
58     'max_features': [2],
59     'min_samples_leaf': [4],
60     'min_samples_split': [2],
61     'n_estimators': [100, 200, 500, 1000, 1500]
62 }
63
64 gridF = GridSearchCV(estimator = rf, param_grid = param_grid, cv = 3,
65     n_jobs = -1, verbose = 2, scoring='accuracy')
66 bestF = gridF.fit(train_data, y_train1)
67
68 # afficher les param tres optimaux
69 print("Les parametre optimaux pour le Random forest sont {} avec un
70     score de {:.2f}".format(bestF.best_params_, bestF.best_score_))
71
72 #RECHERCHE DES HYPER PARAMETRE 2 AVEC HPSKLEARN
73 #on est obliger de transformer les Dataframe en numpy arrays pour que
74 cette methode marche
75 test_datanp = test_data.to_numpy()
76 train_datanp = train_data.to_numpy()
77 y_train1np = y_train1.to_numpy()
78
79 #On utilise HyperoptEstimator de hpsklearn
80 from hpsklearn import random_forest
81 from hpsklearn import HyperoptEstimator
82 from hyperopt import tpe
83 model = HyperoptEstimator( classfier = random_forest('rf.random_forest
84     '),
85
86     algo=tpe.suggest,
87     max_evals=5,
88     trial_timeout=500)
89
90 model.fit(train_datanp,y_train1np)
91
92 # afficher les param tres optimaux

```

```

89 print(model.best_model())
90
91 #comme ont a trouve aucune amelioration avec les hyperparametres ont a
pris la decsion de ne pas en mettre (c'est ce qui nous a donnez le
meilleur score)
92
93 rf = RandomForestClassifier()
94 modelOpt = rf.fit(train_data, y_train1)
95 y_pred = modelOpt.predict_proba(test_data)
96
97 test_transaction = pd.read_csv('test_transaction.csv')
98 y_pred = pd.DataFrame(y_pred)
99 df1 = pd.DataFrame(y_pred, columns = ['isFraud'])
100 df1['isFraud'] = y_pred[1]
101 test1 = test_transaction["TransactionID"]
102 df2 = pd.DataFrame(data=test1, columns=['TransactionID'])
103 test = pd.concat([df2, df1], axis=1)
104 test.to_csv(r'C:\Users\maubr\Desktop\Moi\COURS\M1\testRT12.csv', index
    = False)
105
106
107
108
109 fpr_cv_gauss, tpr_cv_gauss, thr_cv_gauss = metrics.roc_curve(y_test,
    y_pred)
110
111 # calculer l'aire sous la courbe ROC du mod le optimis
112 auc_cv_gauss = metrics.auc(fpr_cv_gauss, tpr_cv_gauss)
113
114 # cr er une figure
115 fig = plt.figure(figsize=(20, 20))
116
117 # afficher la courbe ROC du mod le optimis
118 plt.plot(fpr_cv_gauss, tpr_cv_gauss, '-', lw=2, label='AUC=%.2f' % \
119     ( auc_cv_gauss))
120
121
122 # donner un titre aux axes et au graphique
123 plt.xlabel('False Positive Rate', fontsize=30)
124 plt.ylabel('True Positive Rate', fontsize=30)
125 plt.title('SVM ROC Curve for a gaussian kernel', fontsize=30)
126
127 # afficher la l gende
128 plt.legend(loc="lower right", fontsize=25)
129
130 # afficher l'image
131 plt.show()
132
133 fig = plt.figure(figsize=(20, 20))
134 plot_confusion_matrix(forestOpt, X_test, y_test)
135 plt.title('confusion matrix for a gaussian kernel', fontsize=25)
136 plt.show()

```

Annexe D

Figure Random Forest

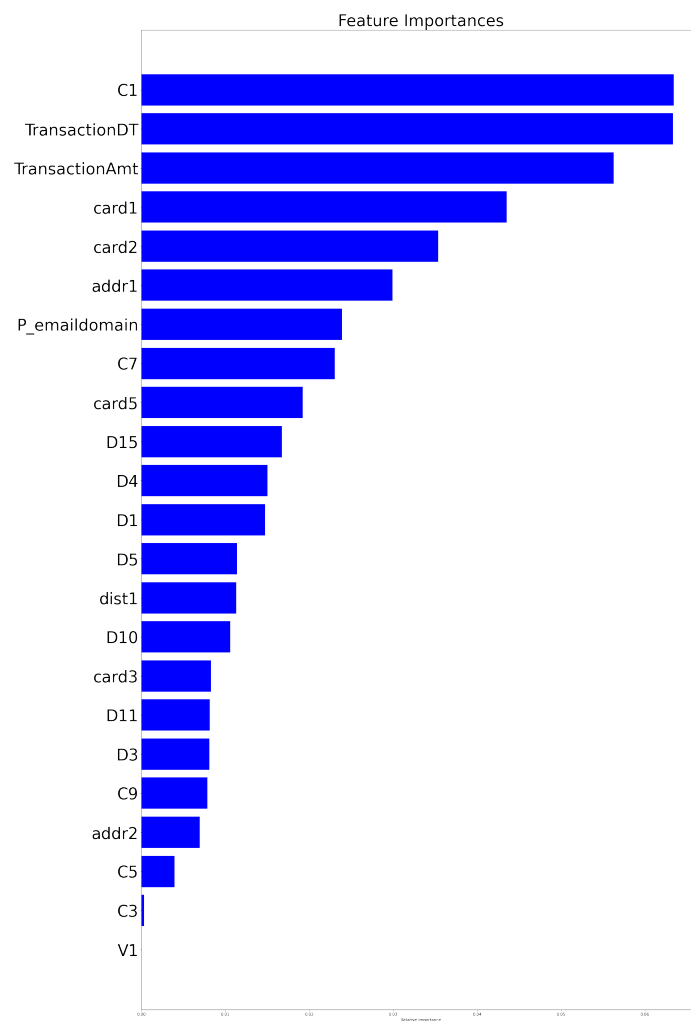


FIGURE D.1 – Variables les plus significatives pour le modèle

Bibliographie

- [1] Explication de la Base de Donnée,
<https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203>
- [2] Exemple de solution pour le probleme de Fraud Detection,
<https://www.kaggle.com/jonas0/ieee-fraud-detection-tutorial>
- [3] Random Forest site,
<https://blog.ysance.com/algorithm-n2-comprendre-comment-fonctionne-un-random-forest-en-5-min>
<https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6>
- [4] SVM site,
<https://openclassrooms.com/fr/courses/4470406-utilisez-des-modeles-supervises-non-lineaires/4722466-classifiez-vos-donnees-avec-une-svm-a-noyau>
- [5] Les principes de l'undersampling et de l'oversampling,
https://medium.com/@bluedme_tech/comment-traiter-les-problèmes-de-classification-déséquilibrée-en-machine-learning-8c3bc95ca25b
- [6] Librairies Sklearn,
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- [7] Recherche des hypers Paramètres,
<https://distill.pub/2020/bayesian-optimization/>
<https://towardsdatascience.com/hyperparameter-optimization-in-python-part-2-hyperopt-5f661db91324>
<https://www.hindawi.com/journals/complexity/2019/6278908/>
- [8] Le Cross Validation,
<https://machinelearningmastery.com/nested-cross-validation-for-machine-learning-with-python/>
<https://nycdatascience.com/blog/student-works/fraud-detection-with-vesta-corporation/>