Université de Cergy-Pontoise

Rapport de projet Data Mining

Income Census

Rapporteur :
Tao-Yuan JEN

Réalisé par :
Hong Phuc VU
Loïc TRAMIS

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Description de dataset

Les données présentées dans le dataset sont extrait de la base de données 1994 Census bureau par Ronny Kohabu et Barry Becker. L'ensemble de données pertinents sont extraites en respectant les conditions suivantes :
- AAGE > 16
- AGI > 100
- AFNLWGT > 1
- HRSWK > 0

Le fichier zip de dataset contient 3 fichiers suivants
- adult.data : Données de salaire pour entraîner le modèle de prédiction
- adult.test : Données de salaire pour tester le modèle de prédiction
- adult.test : Description de dataset

48842 instances, mixte de valeur continue et discrète (train=32561, test=16281)
Probabilités de la classe de fichier adult.all
Probabilité pour le label '>50K' : 23.93%
Probabilité pour le label '<=50K' : 76.07%

1.2 But de projet

Dans le cadre de l'analyse de données, on va concevoir un modèle de prédiction pour prédire si la personne gagne plus de 50K dollars annuel en fonction de informations disponible de chaque individuel dans les données de salaire.

1.3 Techniques

Pour résoudre les pronlème présentés dans le dataset, on utilisera les approches de construction de modèle de prédiction de Machine Learning. Plus concrètement, on va s'appuie sur 3 méthodes de classification le plus connues.
- Arbre de décision
- Forêts Aléatoires
- K-Nearest Neighobors

1.4 Déroulement

La démarche se déroule comme suit :
- Analyse de données
- Feature Engineering
- Arbre de décision
- Forêts Aléatoires
- K-Nearest Neighobors
- Conclusion

Chapitre 2

Analyse de données

2.1 Caractéristique statistique de dataset

Aperçu de données

On utilise la commande str() dans R pour vérifier les caractéristiques de chaque colonne. Le résultat est comme suit

```
'data.frame': 32561 obs. of  15 variables:
 $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
 $ workclass     : Factor w/ 9 levels "?","Federal-gov",..: 8 7 5 5
    5 5 5 7 5 5 ...
 $ fnlwgt        : int  77516 83311 215646 234721 338409 284582
    160187 209642 45781 159449 ...
 $ education     : Factor w/ 16 levels "10th","11th",..: 10 10 12 2
    10 13 7 12 13 10 ...
 $ education_num : int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital_status: Factor w/ 7 levels "Divorced","Married-AF-spouse
    ",..: 5 3 1 3 3 3 4 3 5 3 ...
 $ occupation    : Factor w/ 15 levels "?","Adm-clerical",..: 2 5 7
    7 11 5 9 5 11 5 ...
 $ relationship  : Factor w/ 6 levels "Husband","Not-in-family",..:
    2 1 2 1 6 6 2 1 2 1 ...
 $ race          : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5
    5 3 3 5 3 5 5 5 ...
 $ sex           : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1
    1 2 1 2 ...
 $ capital_gain  : int  2174 0 0 0 0 0 0 0 14084 5178 ...
 $ capital_loss  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hours_per_week: int  40 13 40 40 40 40 16 45 50 40 ...
 $ native_country: Factor w/ 42 levels "?","Cambodia",..: 40 40 40
    40 6 40 24 40 40 40 ...
 $ salary        : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1
    2 2 2 ...
```

La base de données de salaire consiste en 32561 lignes et 15 colonnes

Il existe deux types de variables catégorique (Factor) et numérique (int).

Les variables de nature numériques sont les suivantes :
age, flnwgt, education_num, capital_gain, capital_loss, hours_per_week.
Le reste de variables s'agit les variables catégoriques :
workclass, education, marital_status, occupation, relationship, race, sex, native_country et salary.
La classe label est la variable salary.

Certains variables ne sont pas évident à interpréter. La variable fnlwgt représente le pois final qui est la nombre d'unité dans la cible population. La variable education_num est le nombre d'années d'étude qui est une représentation continuée de la variable discrète education. La variable relationship représente le rôle correspondant dans la famille. capital_gain et capital_loss sont la salaire des sources d'investissement.

Nous vérifions ensuite les valeurs null affiché NaN en R présenté la même chose que ?

```
colSums(is.na(df))
```

On obtient la réponse suivante

```
age     0
workclass       0
fnlwgt      0
education       0
education_num       0
marital_status      0
occupation      0
relationship        0
race        0
sex     0
capital_gain        0
capital_loss        0
hours_per_week      0
native_country      0
salary      0
```

La commande permet de vérifier les valeurs null présentés dans toutes les colonnes. Il n'y a pas de valeur autre que 0 dans le résultat, alors cela assure que ce dataset contient pas de valeur null.

On s'occupera le manque d'information dans la suite de ce travaux.

2.2 Exploration de variables catégoriques

2.2.1 La variable salary

On utilise la fonctionne count() pour s'afficher les valeurs et sa fréquence. Le résultat ci-dessus est la statistique de variables salary.

```
x freq
<=50K 24720
>50K  7841
```

Il y a deux levels : <=50K, >50K. La classification devient alors binaire. La distribution de la valeur <=50K est 24720 soit 75,92%, celle de la valeur >50K est 7841 soit 24,08%

Pour mieux comprendre la distribution, on l'affiche dans le graphe suivant :

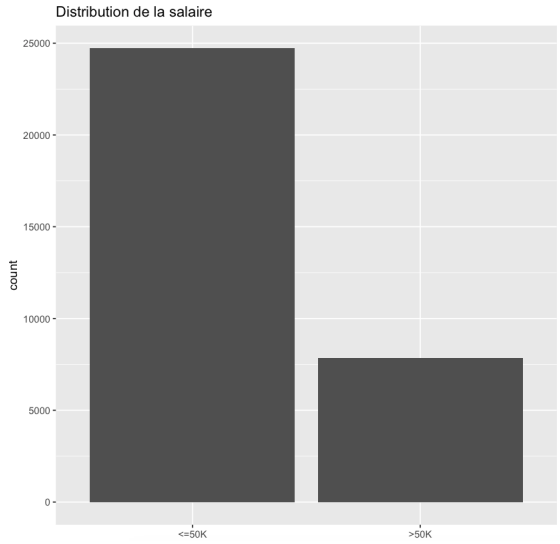

FIGURE 2.1: Distribution de salary

Visualisation de salary par rapport à sex
La photo ci-dessus représente le graphe en R pour la distribution de salaire/sex

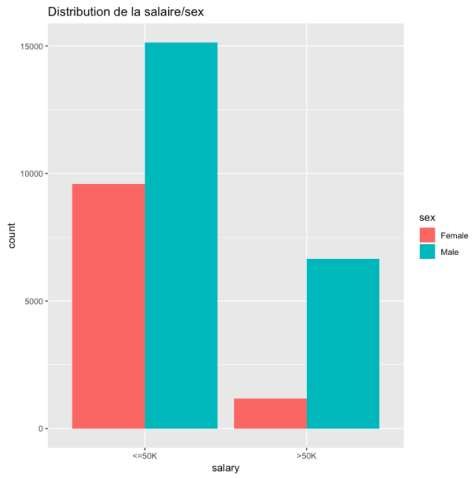

FIGURE 2.2: Distribution de salary/sex

On voit que les hommes gagnent plus que les femmes dans les deux catégories de salaire

Visualisation de salary par rapport à race
La visualisation suivante est la distribution salaire/race

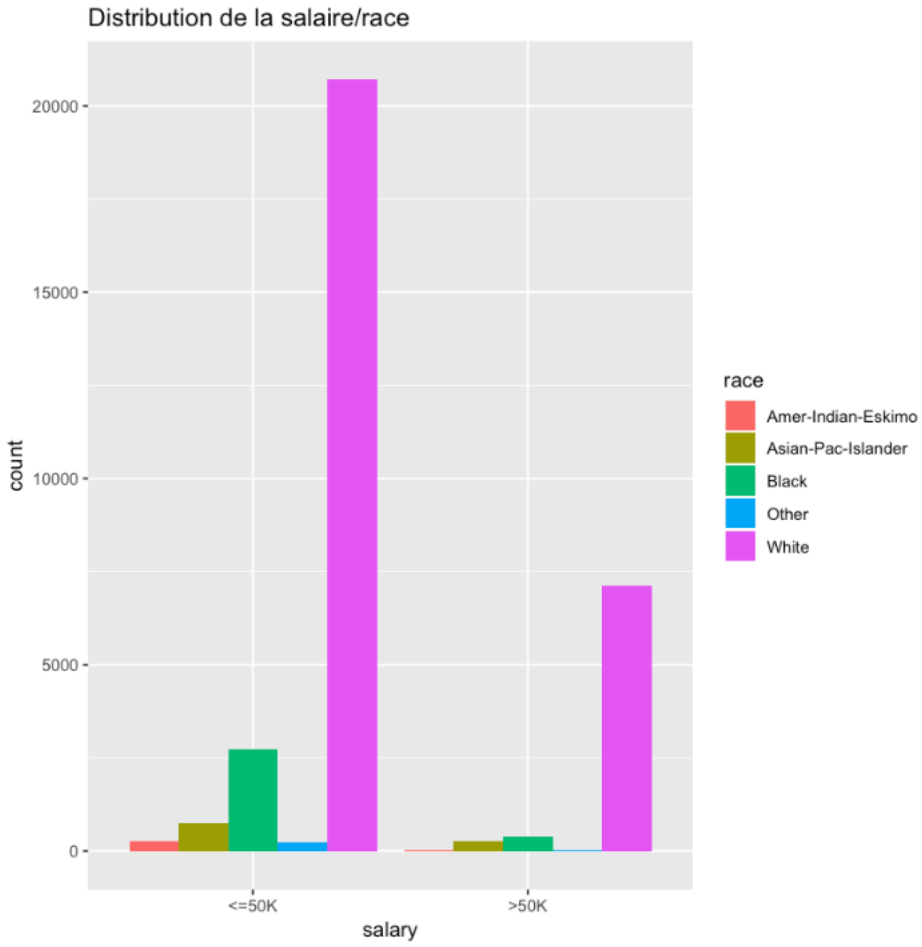

FIGURE 2.3: Distribution de salary/race

On peut observer que les blanc gagnent plus les autres dans les deux catégories

2.2.2 La variable workclass

En utilisant la fonction count, on obtient le tableau de valeurs workclass comme suit

```
x freq
? 1836
Federal-gov 960
Local-gov 2093
Never-worked  7
Private 22696
```

```
Self-emp-inc  1116
Self-emp-not-inc  2541
State-gov 1298
Without-pay 14
```

On trouve dans le tableau 1836 de données non-catégorisé encodé ? pour la variable workclass. Ce sont des données manquées à cause de différents raisons. Sous R, il vaut mieux de convertir ces valeurs à valeur null pour appliquer les techniques de traitement de Data Cleaning. Cette application sera utilisé pour autre variables catégoriques. Le code R est comme ci-dessus

```
df$workclass[ df$workclass == "?" ] <- NA
df$workclass = factor(df$workclass)
```

Visualisation de workclass

Pour valoriser les données de tableau, on les affiche dans le graphe suivant

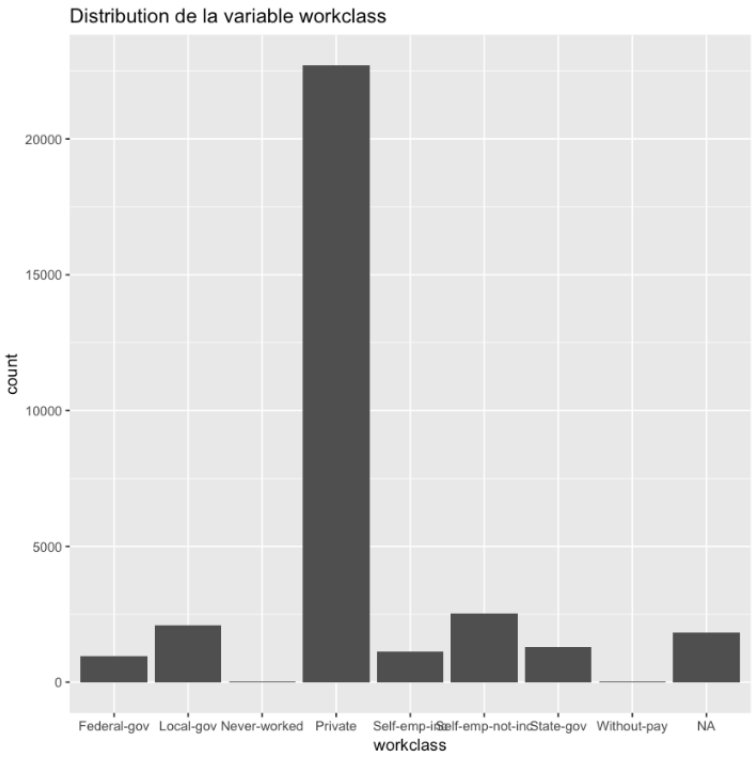

FIGURE 2.4: Distribution de workclass

On constate qu'il y a plus d'employé privé que d'autre genre de type d'emploi.

Visualisation de workclass par rapport à salary

La photo ci-dessus illustre la distribution workclass/salaire

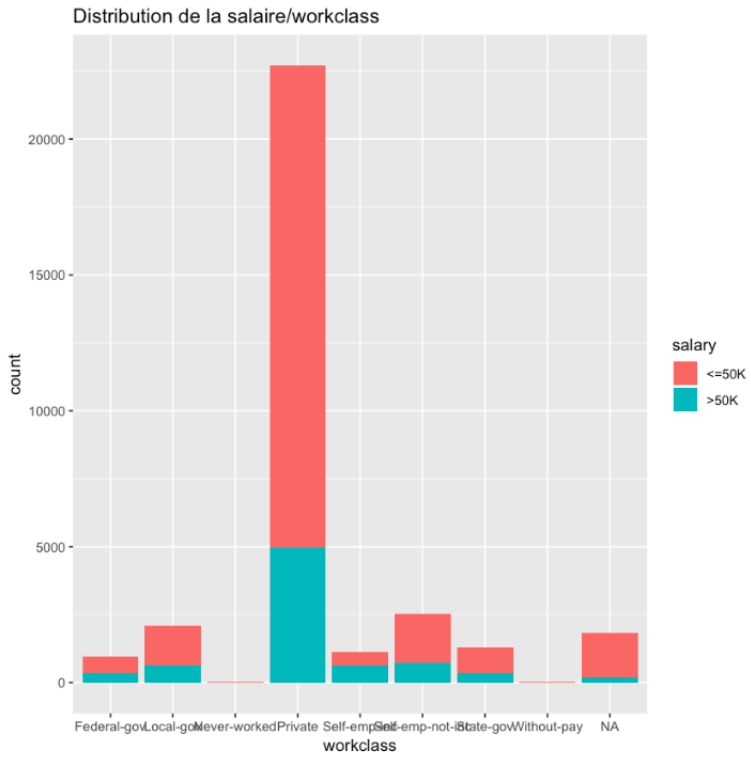

FIGURE 2.5: Distribution de workclass/salary

2.2.3 La variable native_country

Le graphe de distribution de valeurs de native_country est le suivant

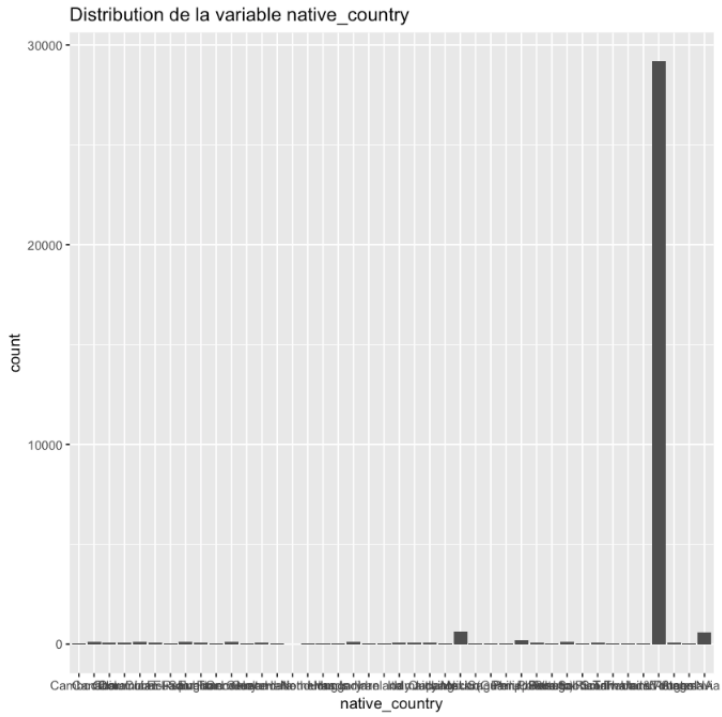

FIGURE 2.6: Distribution de native_country

On voit sur le graphe que le résultat est visiblement très biaisé pour les Etat-Unis. On supprime donc cet attribut de notre dataset.

2.2.4 La variable occupation

Grâce à la fonction dans R, on aura le résultat de valeurs de la variable occupation suivant

```
x freq
Adm-clerical  3770
Armed-Forces  9
Craft-repair  4099
Exec-managerial 4066
Farming-fishing 994
Handlers-cleaners 1370
Machine-op-inspct 2002
Other-service 3295
Priv-house-serv 149
Prof-specialty  4140
Protective-serv 649
Sales 3650
Tech-support  928
```

```
Transport-moving   1597
? 1843
```

Le problème de manque d'information persiste aussi dans la variable occupation. On applique la méthode de conversion en null pour les données possédant la valeur ? 1843 en occurrence.

Visualisation de occupation par rapport à salary
Puis il suffit de visualiser les données occupation sous forme de graphe.

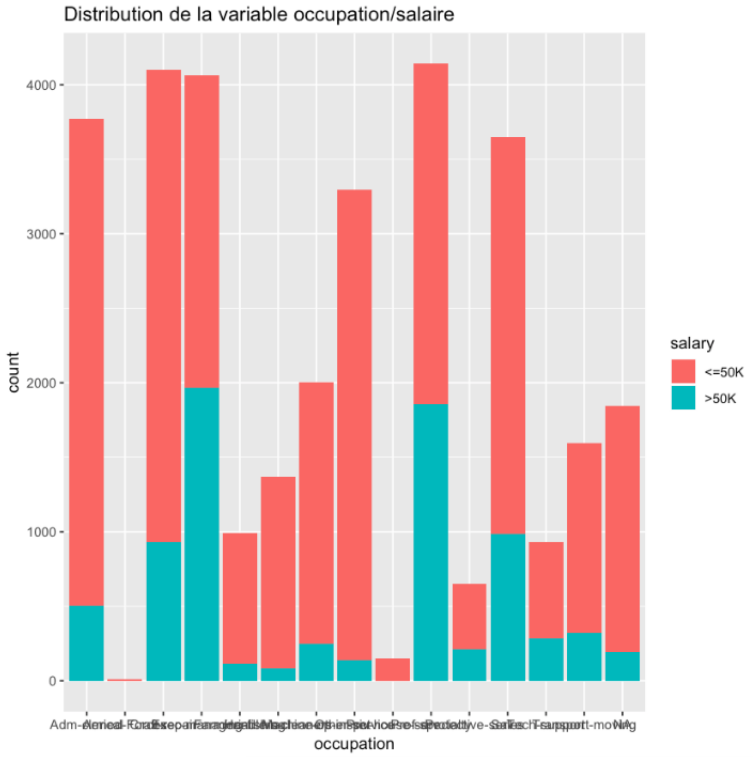

FIGURE 2.7: Distribution de occupation/salary

2.3 Exploration de variables numériques

2.3.1 La variable âge

Visualisation de age par rapport à salary

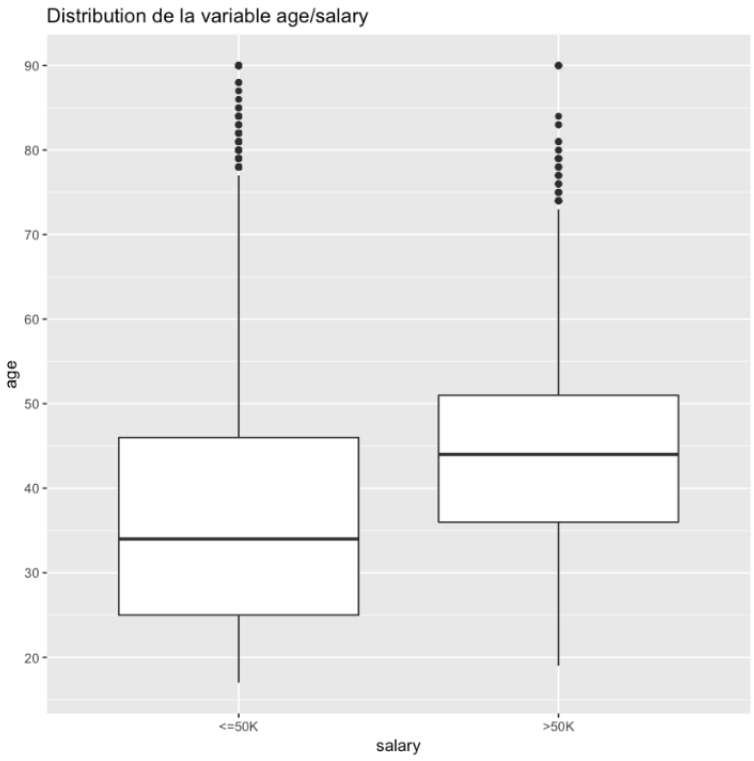

FIGURE 2.8: Distribution de age/salary

On remarque que la majorité les gens qui gagent plus de 50K sont les personnes agés (35-50) alors que les jeunes (25-35) gagent en générale moins de 50K

2.3.2 La variable capital_gain

Visualisation de capital_gain par rapport à salary

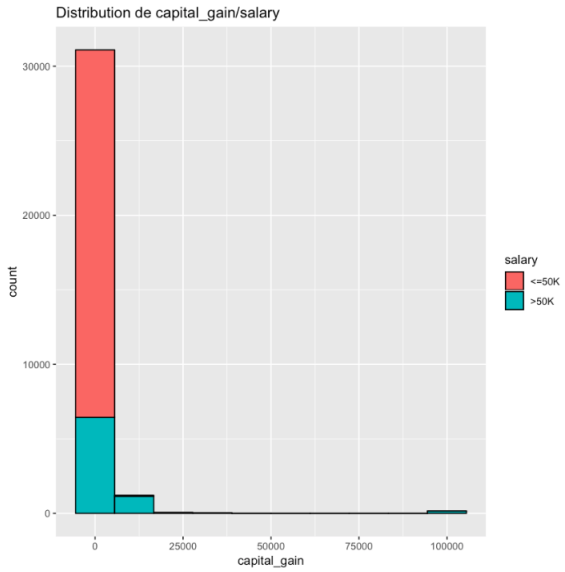

FIGURE 2.9: Distribution de capital_gain/salary

La distribution de capital_gain par rapport à la salaire est extrêmement biaisé vers la valeur 0. Par conséquent, il est logique d'éliminer la variable capital_gain de notre dataset.

2.3.3 La variable capital_loss

Visualisation de capital_loss par rapport à salary

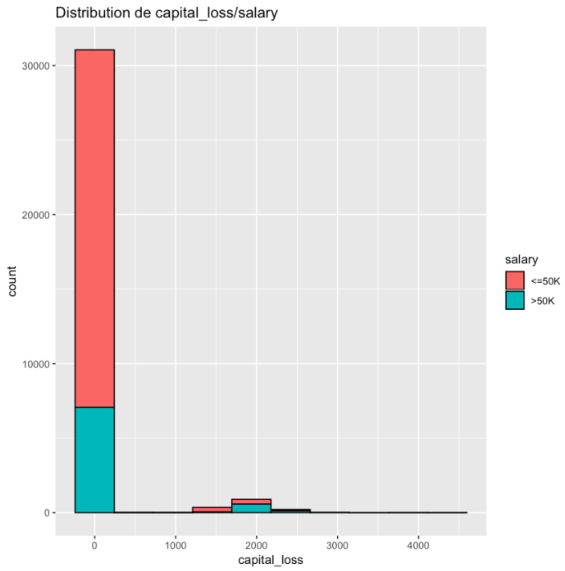

FIGURE 2.10: Distribution de capital_loss/salary

La distribution de capital_loss par rapport à la salaire est également très biaisé vers la valeur 0. On peut donc enlever la variable capital_loss.

Chapitre 3

Feature Engineering

Feature Engineering est le procès de transformation des données bruites en données pertinents. Cela nous permet de mieux comprendre le modèle et d'améliorer la performance de prédiction.

3.1 Remplissage de données manquées

Parmi les attributs du jeu de données, les données manquées autrement dit les valeurs null sont dans certaines variables : workclass, occupation, native_country. On peut facilement vérifier ce fait par la fonction is.na()

Il est nécessaire de traiter les données manquées avant de continuer à appliquer les techniques de Machine Learning parce que les données mal traitées peuvent éventuellement fausser le modèle qu'on essaie de concevoir pour la prédiction de sortie. Le fait que les variables impliquées ci-dessus sont de sorte catégorique, il est possible de traiter avec deux approches :
- Ignorer les lignes contenant les données manquées
- Remplacer par la valeur plus fréquenté dans la colonne d'attribut

Techniquement, on peut choisir la première méthode pour enlever complètement les lignes tronquées vu que son pourcentage n'est pas important soit % de données disponibles. Pourtant, on s'intéresse plutôt à la méthode de remplissage de données le plus fréquent pour les raisons d'étude et de découverte dans le cadre de ce projet. En plus, on se trouve souvent dans la production des datasets avec laquelle les données manquées sont beaucoup plus important. C'est donc une bonne pratique d'appliquer cette méthode.

La méthode de remplissage de données plus fréquent consiste en chercher la valeur la plus dominant de la variable. Il suffit ensuite de remplir tous les champs null par la valeur trouvée à la suite de la recherche. Le code est comme suit

```r
df$workclass[is.na(df$workclass)] <- names(which.max(table(
    df$workclass)))
df$occupation[is.na(df$occupation)] <- names(which.max(table(
    df$occupation)))
```

3.2 Label encoding

Pour certaines algorithmes de Machine Learning tel que celle de regression et clustering, il faut convertir les variables catégoriques en numériques. Cela permet des algorithmes dont prennent en comptes que les valeurs numériques de lire les valeurs catégoriques. L'autre intérêt pour nous, cela permet aussi de construire la matrice de corrélation pour les datasets possédant des variables catégoriques/numériques à la fois.

Les données encodées en valeurs numériques est comme ci-dessus

age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	hours_per_week	salary
39	7	77516	10	13	5	1	2	5	2	40	1
50	6	83311	10	13	3	4	1	5	2	13	1
38	4	215646	12	9	1	6	2	5	2	40	1
53	4	234721	2	7	3	6	1	3	2	40	1
28	4	338409	10	13	3	10	6	3	1	40	1
37	4	284582	13	14	3	4	6	5	1	40	1

FIGURE 3.1: Label encoding

3.3 Corrélation

La matrice de corrélation nous permet de vérifier la relation parmi les variables. Cela surligne surtout le redondance de données dans le jeu de données par exemple les valeurs présentées par les différents type d'unité. On réalise alors un heatmap de corrélation comme ci-dessus.

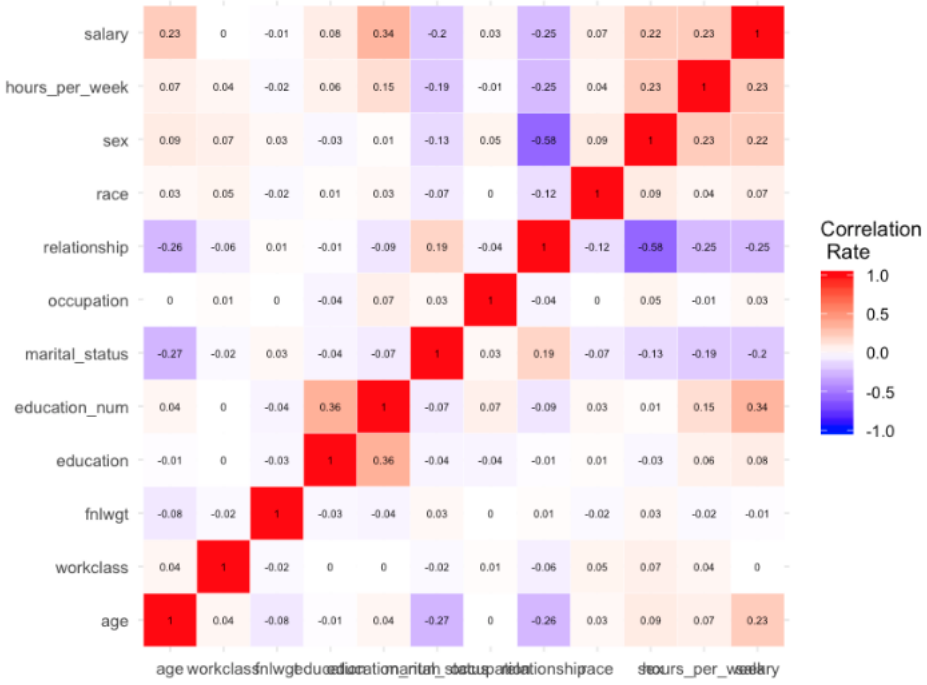

FIGURE 3.2: Heatmap de corrélation

Comme on peut voir dans la visualisation les deux variables education et educations_num sont fortement corrélées. En effet, ses valeurs sont identiques et symétriques sur la matrice. on avait le même phénomène pour le couple de variable relationship et martial_status. Par conséquent, on pourra enlever les variables relationship et education. Il est préférable de garder les variables numériques pour construire le modèle.

3.4 One hot encoding

La méthode de label encoding s'avère efficace pour construire la matrice de corrélation. Par contre, elle assume qu'il existe une relation ou bien une ordre entre les valeurs d'une variable alors qu'ils sont tous indépendants. Ce sera un problème pour l'application de modèle KNN qu'on souhaite utiliser. De ce fait, il suffit d'utiliser un autre technique : One hot encoding

One hot encoding permet d'éclater une colonne qui contient les valeurs catégoriques en multiples colonnes dépend de nombre de valeurs disponibles. Ensuite, il remplit la valeur par 1 si la colonne correspondant à la valeur avant le découpage, 0 sinon. L'application de one hot encoding sur les données :

	age	workclass.Federal.gov	workclass.Local.gov	workclass.Never.worked	workclass.Private	workclass.Self.emp.inc	workclass.Self.emp.not.inc	workclass.State.gov	workclass.Without.pay	fnlwgt	...	occupation.Tech.support	occupation.Transport.moving
23509	24	0	0	0	1	0	0	0	0	219510	...	0	0
19656	32	0	0	0	1	0	0	0	0	207668	...	0	0
24705	60	0	0	0	1	0	0	0	0	139391	...	0	0
9017	56	0	0	0	1	0	0	0	0	132026	...	0	0
15969	38	0	0	0	1	0	0	0	0	275223	...	0	0
761	36	0	0	0	1	0	0	0	0	101460	...	0	0

FIGURE 3.3: One hot encoding

3.5 Normalisation de données

L'algorithme K-NN nécessite que les données présentées sont sur la même échelle. En conséquence, il faut normaliser ces données, on utilise la méthode de normalisation min-max dans le cadre de ce projet. La fonction est décrit comme ci-dessus

```
MinMaxScaling <- function(x){
  return((x-min(x))/(max(x)-min(x)))
}
```

3.6 Coupage de données train/test

On sépare les données de dataset en deux set : train et test. Le set train permet de d'apprendre le modèle et le set test permet de tester la performance de notre modèle conçu. La taille de découpage est 70/30 compris 70% de données train et 30% de données test initialement. On va varier éventuellement ce proportion pour trouver le modèle le plus performant.

Pour chaque partie, on regroupe les attributs dans une variable X et la cible dans une variable y. La convention pour le set train est comme suit
- X_train : la variable pour le tableau de variables entrées du modèle
- y_train : la variable pour le vecteur de label du modèle

Chapitre 4

Arbre de décision

4.1 Arbre standard sans paramétrage

L'arbre de décision aide à la décision ou d'exploration de données permet de représenter un ensemble de choix sous la forme graphique d'un arbre. C'est une des méthodes d'apprentissage supervisé les plus populaires pour les problèmes de classification de données.

Concrètement, un arbre de décision modélise une hiérarchie de tests pour prédire un résultat. Il existe deux principaux types d'arbre de décision :
- Les arbres de régression (Regression Tree) permettent de prédire une quantité réelle, une valeur numérique
- Les arbres de classification (Classification Tree) permettent de prédire à quelle classe la variable de sortie appartient

Avantages

Les arbres de décision ont pour avantage d'être simple à interpréter, très rapide à entraîner, d'être non paramétrique, et de nécessiter très peu de prétraitement des données. Ils peuvent être calculés automatiquement par des algorithmes d'apprentissage supervisé capables de sélectionner automatiquement les variables discriminantes au sein de données non-structurées et potentiellement volumineuses. Ces algorithmes permettent aussi d'extraire des règles logiques qui n'apparaissaient pas dans les données brutes.

Mise en place

La classification par l'arbre de décision se décompose en deux étape : apprentissage et classification. On commence à construire le modèle sans paramètre pour avoir une idée sur la performance avec les paramètre choisi par défaut.

On utilise la fonction rpart pour construire l'arbre. Il faudra cependant de préciser la source de données, la cible (label) et la méthode. Le set train est utilisé pour l'entraînement du modèle. La source est donc X_train et la cible à apprendre est y_train. La syntaxe est comme suit.

```
1 treeModel <- rpart(y_train~., data=X_train, method="class")
```

Une fois la phase d'apprentissage est fini, on est capable de visualiser l'arbre du modèle par la fonction plot. L'arbre pour classifier le dataset est comme la photo ci-dessus.

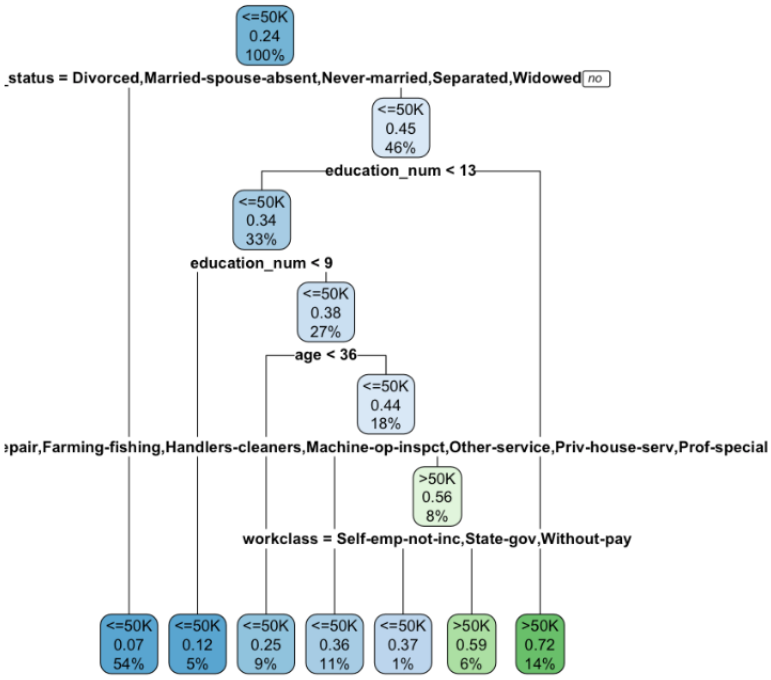

FIGURE 4.1: Arbre de décision standard

Il est moment de tester l'arbre déduit par le set train. On applique la fonction predict sur le set test avec le modèle d'apprentissage qu'on obtient et les attributs en entrées venu de set test.

```
1 pred <- predict(model, feature_test, type = "class")
```

Le résultat estimé par la fonction predict sera utilisé pour comparer à des vraie label. Cette comparaison nous permet de créer la matrice de confusion qui contient les informations pertinents pour déduire la performance de notre

modèle. On obtient à la suite de notre testing la matrice de confusion ci-dessus.

```
1         target_test
2 pred     <=50K >50K
3   <=50K   6844 1101
4   >50K     550 1274
```

pred s'agit de label prédit et target_test contient le vrai label. On s'interprète cette matrice base sur le tableau de convention ci-dessus.

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

FIGURE 4.2: Matrice de confusion

Quant à notre matrice de confusion, on a donc 6844 vrai positif, 1274 vrai négatif, 1101 faux positif et 550 faux négatif

A partir de la matrice de confusion, il existe des formules pour déterminer les différents indicateurs de performance. On s'intéresse surtout à l'indicateur accuracy qui permet de mesurer la précision de modèle d'apprentissage. On pourrait calculer aussi autres indicateurs via la formule suivante.

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

precision or positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$

negative predictive value (NPV)

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} = 1 - \mathrm{FOR}$$

FIGURE 4.3: Formule de calcul d'indicateurs

Pour faciliter le calcul de ces faits, il est préférable d'écrire une fonction dans R pour faire le travail. Le code est comme ci dessus

```
statMC <- function(mc) {
    print(mc[1,2])
    accuracy <- sum(diag(mc)/(sum(rowSums(mc)))) * 100
    sensitivity <- mc[1,1]/(mc[1,1] + mc[2,1]) #True positive rate
    specificity <- mc[2,2]/(mc[1,2] + mc[2,2])
    pos_pred <- mc[1,1]/(mc[1,1] + mc[1,2])
    neg_pred <- mc[2,2]/(mc[2,1] + mc[2,2])
    #arithmetic_mean <- (recall + true_negative)/2
    #geometric_mean <- sqrt(recall * true_negative)

    cat("Accuracy: ", accuracy, "\n")
    cat("Sensitivity: ", sensitivity, "\n") #Precision nombre de
    positive bien classifi
    cat("Specificity: ", specificity, "\n") #nombre de n gative
    bien classifi
    cat("Pos Pred Value: ", pos_pred, "\n")
    cat("Neg Pred Value: ", neg_pred, "\n")

}
```

Finalement le modèle d'abre de décision avec les paramètres par défaut donne accuracy de 83.1%. Ce résultat est vraiment respectable considéré qu'on fait aucun paramétrage dessus.

```
Accuracy:  83.0996
Sensitivity:  0.9256154
Specificity:  0.5364211
Pos Pred Value:  0.8614223
Neg Pred Value:  0.6984649
```

4.2 Arbre tunning

4.2.1 Pre-pruning

L'une des questions qui se posent dans un algorithme d'arbre de décision est la taille optimale de l'arbre final. Un arbre trop gros risques surapprentissage les données de formation et généralisent peu à de nouveaux échantillons. Un petit arbre pourrait ne pas saisir d' importantes informations structurelles sur l'espace de l'échantillon. Pour éviter ces probleme, on pourrait adopter le technique l'élagage des arbres de décision. Ce technique permet de réduit la taille des arbres de décision en supprimant les sections de l'arbre qui fournissent peu de pouvoir classifier les instances. La taille réduit la complexité de la finale classificateur et améliore donc la précision prédictive par la réduction des overfitting.

Le tunning de paramètre de l'arbre par le coupage nous donne un accuracy un peu mieux qu'avant. On met 8 comme valeur maximale pour la profondeur

et le nombre de split minimum à 100. Le paramétrage est comme suit

```
treeModelPre <- rpart(y_train~., data = X_train, method = "class",
                      control = rpart.control(cp = 0, maxdepth = 6,
    minsplit = 100))
```

Une visualisation de l'arbre pre-pruning

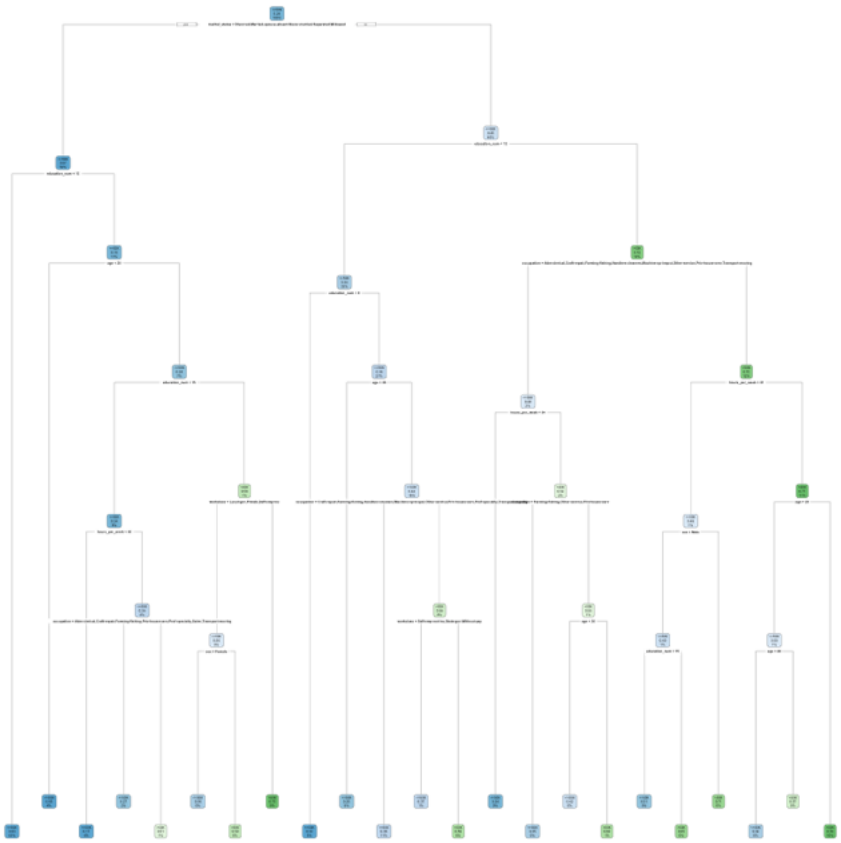

FIGURE 4.4: Arbre pre-pruning

L'accuracy obtenu est 83.46%

```
Accuracy:  83.45788
Sensitivity:  0.9350825
Specificity:  0.5216842
Pos Pred Value:  0.858882
Neg Pred Value:  0.7207679
```

Chapitre 5

Forêts Aléatoires

5.1 Forêts Aléatoires

Dans la famille de classification, on trouve aussi la méthode de Random Forest (forêts aléatoires). L'algorithme des forêts aléatoires est un algorithme de classification qui réduit la variance des prévisions d'un arbre de décision seul, améliorant ainsi leurs performances. Pour cela, il combine de nombreux arbres de décisions dans une approche de type bagging. Il effectue un apprentissage en parallèle sur de multiples arbres de décision construits aléatoirement et entraînés sur des sous-ensembles de données différents. Le nombre idéal d'arbres, qui peut aller jusqu'à plusieurs centaines voire plus. Concrètement, chaque arbre de la forêt aléatoire est entrainé sur un sous ensemble aléatoire de données selon le principe du bagging, avec un sous ensemble aléatoire de features (caractéristiques variables des données) selon le principe des projections aléatoires. Les prédictions sont ensuite moyennées lorsque les données sont quantitatives ou utilisés pour un vote pour des données qualitatives, dans le cas des arbres de classification.

Avantage

L'algorithme des forêts aléatoires est connu pour être un des classifieurs les plus efficaces out-of-the-box (c'est-à-dire nécessitant peu de prétraitement des données).
- Elles donnent de bons résultats surtout en grande dimension
- Elles sont très simples à mettre en œuvre
- Elles ont peu de paramètres.

Mise en place

On commence à effectuer le modèle par la fonction randomForest. Les paramètres ressembles à celui de l'arbre de décision classique. On a quand même

un nouveau paramètre importance

```
modelRF = randomForest(y_train~., data = X\_train, importance =
    TRUE)
```

Sans préciser les paramètres, le nombre d'arbre pour faire marcher l'algorithm est 500, le nombre de variables minimum prise pour le découpage est 3

Cette fois on obtient un modèle avec accuracy nettement mieux par rapport à CART standard de 83.57%

```
Accuracy:  83.57048
Sensitivity:  0.9152015
Specificity:  0.5882105
Pos Pred Value:  0.873725
Neg Pred Value:  0.6902174
```

Chapitre 6

K Nearest Neighbors

6.1 K-NN

L'algorithme K-NN (K-nearest neighbors) est une méthode d'apprentissage supervisé. Il peut être utilisé aussi bien pour la régression que pour la classification. Pour effectuer une prédiction, l'algorithme K-NN ne va pas calculer un modèle prédictif à partir d'un Training Set comme c'est le cas pour la régression logistique ou la régression linéaire. En effet, K-NN n'a pas besoin de construire un modèle prédictif. Ainsi, pour K-NN il n'existe pas de phase d'apprentissage proprement dite. Pour pouvoir effectuer une prédiction, K-NN se base sur le jeu de données pour produire un résultat.

6.2 Mise en place

En raison de manque de modèle construit pour K-NN. On précise directement dans les champs de paramètres les données entrées de set train et test ainsi que la sortie (label) de train set. La fonction permet de lancer l'algorithme K-NN est knn. La syntaxe est comme suit

```
tab <- table(pr,y_test)
```

```
Accuracy:  73.4978
Sensitivity:  0.9390046
Specificity:  0.09978947
Pos Pred Value:  0.7645634
Neg Pred Value:  0.3444767
```

On a l'accuracy de 73.49 avec k=10.

6.3 K-NN optimisé

Pour optimiser la performance de K-NN, il nous faut trouver un nombre de voisin le plus adéquate. La performance de prédiction de K-NN dépend fortement sur le choix de voisin. Ainsi, on fait une boucle sur le nombre de voisin de 1 à 28 pour trouver celui qui donne le résultat le plus satisfaisant.

```
1 = 67.33545
2 = 65.81022
3 = 69.99693
4 = 69.94575
5 = 71.66547
6 = 71.645
7 = 72.52534
8 = 72.75054
9 = 73.33402
10 = 73.08834
11 = 74.16317
12 = 74.18364
13 = 74.54192
14 = 74.47026
15 = 74.76712
16 = 74.86949
17 = 75.06398
18 = 75.02303
19 = 75.17658
20 = 75.00256
21 = 75.29942
22 = 75.3506
23 = 75.3506
24 = 75.33013
25 = 75.39154
26 = 75.41202
27 = 75.45296
28 = 75.41202
```

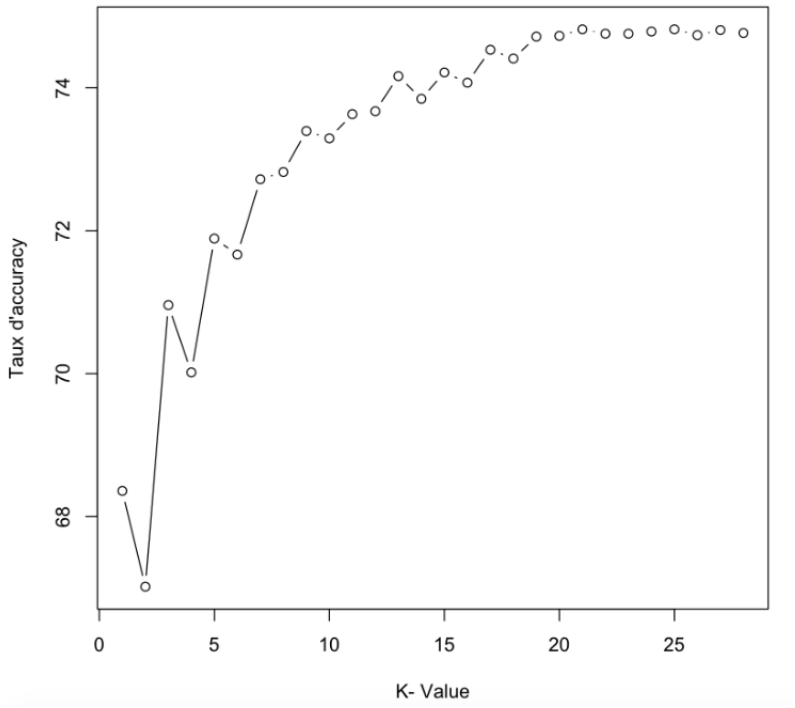

FIGURE 6.1: Taux d'accuracy de K-NN avec k de 1 à 28

On remarque que plus qu'on augmente la valeur de k plus que le résultat améliore. Cependant, un moment le taux n'évolue plus, il y a peu d'amélioration quand k varie de 15 à 28, le résultat est assez stable.

Chapitre 7

Conclusion

7.1 Test de modèle de prédiction

Dans cette section, on s'affiche le résultat de test de tous les modèle conçus tout au long de ce projet sur le set de test adult.test

7.1.1 Arbre de décision

On utilise le modèle pre-pruning qui avait la meilleur performance. Le résultat produit par ce modèle est le suivant

```
Accuracy:  83.2713
Sensitivity:  0.9319934
Specificity:  0.5288971
Pos Pred Value:  0.8582369
Neg Pred Value:  0.7176259
```

Comparé au résultat de 83,46% effectué sur le test set découpé, cela fonctionne moins bien avec le taux de 83.27% mais la performance n'est pas du tout diminué considérablement.

7.1.2 Forêts aléatoires

Le modèle construit par les forêts aléatoires est comme ci-dessus

```
Accuracy:  82.68364
Sensitivity:  0.907563
Specificity:  0.5797985
Pos Pred Value:  0.8685847
Neg Pred Value:  0.6720959
```

On a la même tendance que celle d'arbre de décision, le modèle donne un résultat moins performant que celui obtenu par le test set découpé de données de fichier adult.train, 82.68% contre 83.57%

7.1.3 K-NN

Après avoir testé avec plusieurs nombre de voisins dans la chapitre K-NN, il s'avère que 25 est le nombre k le plus optimal pour faire tourner cet algorithme. Le résultat final est

```
Accuracy:  74.97225
Sensitivity:  0.9907303
Specificity:  0.01219512
Pos Pred Value:  0.7542541
Neg Pred Value:  0.3006536
```

Cette fois, l'algorithme K-NN nous envoie un résultat nettement mieux que celui réalisé pour le test set, 74.97% par rapport à 74.49%

7.2 Conclusion

Après avoir appliqué les algorithme de classification le plus connus, on peut tirer une conclusion c'est que la classification par l'arbre de décision et par les forêts aléatoires marche le mieux pour notre problème existé dans le dataset. Les deux techniques donne les résultats vraiment fiables. En effet, il n'y a pas trop de variance en configurant de paramètres, le taux d'accuracy va jamais diminuer significativement par les mauvais paramètres. De plus, ces deux techniques ont besoin moins de paramétrage et de transformation de données par rapport à K-NN. Ceci est un gros avantage pour traiter les données, problèmes en mode production ou les données sont plus sales et complexe à transformer. D'ailleurs, il y a beaucoup de similarités entre l'arbre de décision et les forêts aléatoires, ça permet de passer de l'un à l'autre sans demander trop d'efforts.

Si on doit choisir une méthode le moins pénible à s'appliquer pour prédire rapidement et efficace sur ce genre de données qu'on a vu dans ce projet, les forêts aléatoires semble un choix idéal. Cela demande aucun paramétrage, il marche out-of-the-box et la capacité de prédiction est vraiment respectable. Cependant si on souhaite passer le temps à tester et à jouer sur le jeu de données pour trouver un modèle le plus performant, le choix d'arbre de décision est évident pour résoudre toute sorte de problèmes.