BELKARFA LINA – AUBESPIN ALBAN

PROJET DATA MINING

Sommaire :

Objectifs du projet :

L'objectif est la création d'un modèle de prédiction du risque de défaut de payement pour les clients et son application aux instances à prédire. On souhaite donc utiliser les techniques de classification afin de générer un modèle de prédiction de la classe des clients :

- *default = Oui (positif)*
- *default = Non (négatif)*

1) Exploration des données et interprétation des résultats

Il est important dans un premier temps d'observer les données dont nous disposons. Dans notre cas, le fichier « Data Projet.csv », bien qu'il soit présenté d'une façon habituelle (séparateur habituel, nom de chaque colonne etc…), il comporte tout de même certaines particularités.
Après avoir stocké nos données dans une variable nommée « Data » :

```
Data <- read.csv("Data Projet.csv", header = TRUE, sep = ",", dec = ".")
```

Nous pouvons faire un « head » de nos données :

```
> head(Data)
  branch ncust customer age            ed employ address income debtinc creddebt
1      3  3017    10012  28         Bac+2      7       2     44    17.7 2.990592
2      3  3017    10017  64 Bac+5 et plus     34      17    116    14.7 5.047392
3      3  3017    10030  40    Niveau bac     20      12     61     4.8 1.042368
4      3  3017    10039  30    Niveau bac     11       3     27    34.5 1.751220
5      3  3017    10071  35    Niveau bac      2       9     38    10.9 1.462126
6      3  3017    10096  26         Bac+3      2       4     38    11.9 0.954142
    othdebt default
1  4.797408     Non
2 12.004608     Non
3  1.885632     Non
4  7.563780     Non
5  2.679874     Oui
6  3.567858     Oui
```

Une chose apparaît de façon évidente, les données sont **triées**, les clients d'une même catégorie comme **« branch »** sont regroupés. Les données doivent être **mélangées** avant d'être séparées en ensemble d'apprentissage et de test. De plus, la variable **« customer »** ne fait pas partie des variables prédictives d'après le dictionnaire de données, c'est un numéro d'identification qui pourrait biaiser les prédictions. Nous pourrons donc la retirer.

Dans un second temps, il faut observer les variables catégorielles utiles :

```
> summary(Data)
     branch          ncust          customer           age                      ed
 Min.   : 3.0   Min.   :1919   Min.   : 10012   Min.   :18.00   Bac+2         :422
 1st Qu.:20.0   1st Qu.:2658   1st Qu.: 98137   1st Qu.:23.00   Bac+3         :268
 Median :64.0   Median :3491   Median :316154   Median :31.00   Bac+4         :248
 Mean   :52.2   Mean   :3478   Mean   :257694   Mean   :34.13   Bac+5 et plus: 70
 3rd Qu.:75.0   3rd Qu.:4358   3rd Qu.:370744   3rd Qu.:41.25   Niveau bac    :192
 Max.   :91.0   Max.   :4809   Max.   :453777   Max.   :79.00
     employ          address           income          debtinc          creddebt
 Min.   : 0.00   Min.   : 0.000   Min.   :  12.00   Min.   : 0.000   Min.   : 0.0000
 1st Qu.: 0.00   1st Qu.: 1.000   1st Qu.:  27.00   1st Qu.: 4.875   1st Qu.: 0.4090
 Median : 3.00   Median : 5.000   Median :  39.00   Median : 8.500   Median : 0.9506
 Mean   : 6.95   Mean   : 6.282   Mean   :  59.96   Mean   : 9.967   Mean   : 1.9460
 3rd Qu.:10.00   3rd Qu.: 9.000   3rd Qu.:  64.00   3rd Qu.:13.600   3rd Qu.: 2.2193
 Max.   :63.00   Max.   :34.000   Max.   :1079.00   Max.   :40.700   Max.   :35.9727
    othdebt        default
 Min.   : 0.000   Non:752
 1st Qu.: 1.111   Oui:448
 Median : 2.221
 Mean   : 3.887
 3rd Qu.: 4.542
 Max.   :63.473
```

Lorsque l'on compare les variables au Dictionnaire de données, on observe que :

-« branch » est une variable devant être considérée comme catégorielle, or elle est considérée comme une variable numérique, comme le montre les médianes, moyenne etc… Nous devrons donc changer cela.

-« ed » est une variable catégorielle qui peut être classées comme ordinale pour plus de précision, il suffit d'y ajouter des **« levels ».**

2) Pré traitements appliqués aux données et description de la méthode de création des données d'apprentissages et de test

Il est maintenant temps de retirer **« customer »,** modifier les paramétrages de **« ed »** et « **branch** ». Ensuite nous mélangeons nos données grâce à « createDataPartition » qui nous permet de choisir le pourcentage choisi ainsi que la variable d'instance.

```
# Importation des données
Data <- read.csv("Data Projet.csv", header = TRUE, sep = ",", dec = ".")
#La variable customer n'est pas pertinente dans l'étude, on la retire
Data<-Data[,-3]
#La variable branch est de type int, nous devons la changer en variable catégorielle
Data$branch<-as.factor(Data$branch)
#La variable ed est de type catégorielle ordinale, on peut indiquer l'ordre des préférences
Data$ed<-factor(Data$ed, ordered = TRUE, levels= c("Niveau bac","Bac+2","Bac+3","Bac+4","Bac+5 et plus"))
# On ajoute une colonne index pour facilité la création de partition aléatoire
Data_with_index <- data.frame(id=1:length(Data$default),Data)
#set.seed pour rendre reproductible les résultats
set.seed(100)
# Index pour l'ensemble d'apprentissage (70%)
index_EA<-createDataPartition(Data_with_index$default, times = 1, p = 0.7,list = FALSE)
# On selectionne uniquement les ligne dont id est dans index_EA
EA_with_index <- Data_with_index[index_EA,]
# On supprime la colonne id
EA<-EA_with_index[-1]
# Idem pour l'ensemble de test en excluant les ligne dont l'index est dans index_EA:
ET_with_index <- Data_with_index[-index_EA,]
# Idem, On supprime la colonne id
ET<-ET_with_index[-1]
```

Pour cela il a fallu rajouter un index pour chaque client qui nous a permis de choisir aléatoirement parmi ces index.

Enfin, une petite vérification du split :

```
# Verification du split:
nrow(EA)/nrow(Data)  # 70% ok
nrow(ET)/nrow(Data)  # 30% ok
print("Data")
print(prop.table(table(Data$default))) #37,3% de 'Oui' dans Data
print("EA")
print(prop.table(table(EA$default))) #37,3% de 'Oui' dans EA
print("ET")
print(prop.table(table(ET$default))) #37,3% de 'Oui' dans ET
#Le split semble être bien réalisé
```

Nos données sont maintenant prêtes à être analysées.

3) Description de la méthode d'évaluation des classifieurs

Nous savons que la classe positive est « default= Oui », et la classe negative « default= Non ».

-default= Oui (positif) -> le client ne rembourse pas

-default=Non (négatif) -> le client rembourse

Ce qui nous intéresse dans chacun des classifieurs, c'est de minimiser le taux de FAUX NEGATIF, c'est à dire minimiser les erreurs de prédictions NEGATIVES (alors qu'elles sont en réalité POSITIVES). De cette manière, on minimise le risque de défaut de paiement. Nous allons donc :

-Minimiser le taux de faux négatif (Défaut de paiement, prédit remboursé)

-Maximiser le taux de vrai négatif (Remboursement de l'emprunt, prédit remboursé)

Nous observerons les taux de succès et surtout les courbes ROC et leurs indices AUC correspondant à leur classifieur, et choisirons le classifieur ayant l'indice de plus élevé. Certains classifieurs seront testés de plusieurs façons (changement de paramètre) et nous choisirons donc ceux possédant les meilleures performances.

4) Description des configurations des classifieurs générés et résultats

a) Classifieur « rpart » (green)

Une des façons par laquelle nous créons le classifieur :

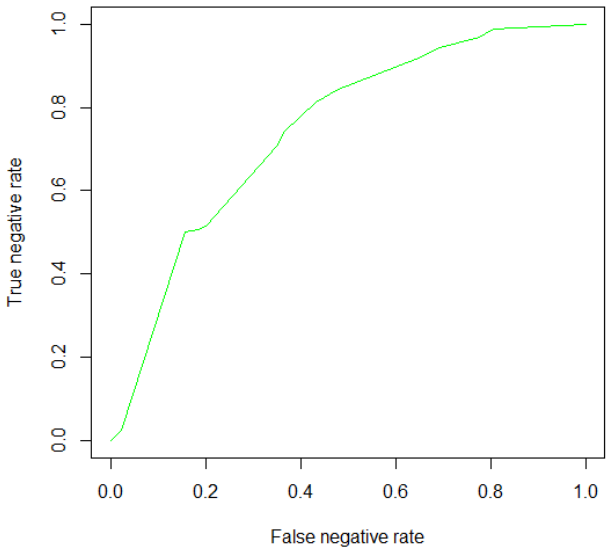
```
#On stock l'arbre construit par l'ensemble d'apprentissage dans Tree1
Tree1 <- rpart(default ~ .,EA)
#On affiche notre arbre (avec prp qui permet une meilleure vision de l'arbre)
prp(Tree1)
#En indiquant le texte correspondant à chaque branche
text(Tree1, pretty = 0)

#Calcul du taux de succès:
#Application de Tree1 à ET
test_Tree1 <- predict(Tree1, ET, type="class")
print(test_Tree1)
table(test_Tree1)

# Stockage des résultats dans un data frame df_Tree1
df_Tree1 <- as.data.frame(table(ET$default, test_Tree1))

# Renommage des colonnes dans le data frame df_tree1
colnames(df_Tree1) = list("Classe", "Prediction", "Effectif")
```

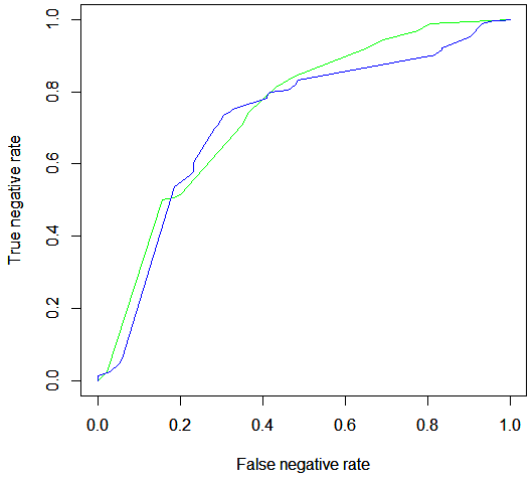
Ensuite on représente graphiquement notre courbe ROC et on calcul le taux de succès et l'AUC :




```
# Calcul de la proportion de succès parmi le nombre total d'exemples de test
sum(df_Tree1[df_Tree1$Classe==df_Tree1$Prediction,"Effectif"])/nrow(ET)
.] 0.7214485
prob_Tree1 <- predict(Tree1,ET, type="prob")
roc_pred_Tree1 <- prediction(prob_Tree1[,2],ET$default)
roc_perf_Tree1 <- performance(roc_pred_Tree1,"tnr","fnr")
plot(roc_perf_Tree1, col = "green")
auc_Tree1 <- performance(roc_pred_Tree1, "auc")
attr(auc_Tree1, "y.values")
.1]]
.] 0.7432504
```

b) Classifieur « C50 » (blue)

On effectue une représentation graphique des deux courbes ROC de « rpart » et « C50 », de cette manière on observe la plus performante.



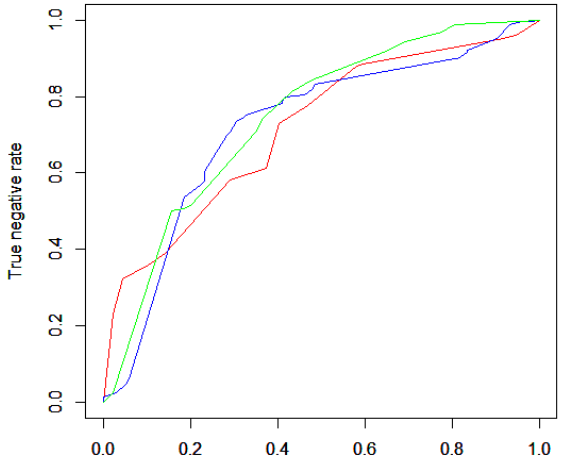
On ne peut pas conclure graphiquement duquel possède la meilleure performance.
C'est pourquoi le taux de succès et l'AUC sont importants :

```
> # Calcul de la proportion de succès parmi le nombre total d'exemples de test
> sum(df_Tree2[df_Tree2$Classe==df_Tree2$Prediction,"Effectif"])/nrow(ET)
[1] 0.7130919
> #INDICE AUC DU 2ND CLASSIFIEUR
> attr(auc_Tree2, "y.values")
[[1]]
[1] 0.7174295
```

On voit ici que l'AUC tout comme le taux de succès est plus élevé dans le classifieur précédent. Ce qui indique qu'il est plus performant et nous intéresse plus dans nos recherches. On ne s'intéresse donc pas au classifieur « C50 ».

c) Classifieur « tree » (red)

On effectue une représentation graphique des trois courbes ROC de « rpart », « C50 », et « tree » de cette manière on observe la plus performante. (On aurait pu retirer C50 car on la sait moins performante que rpart)



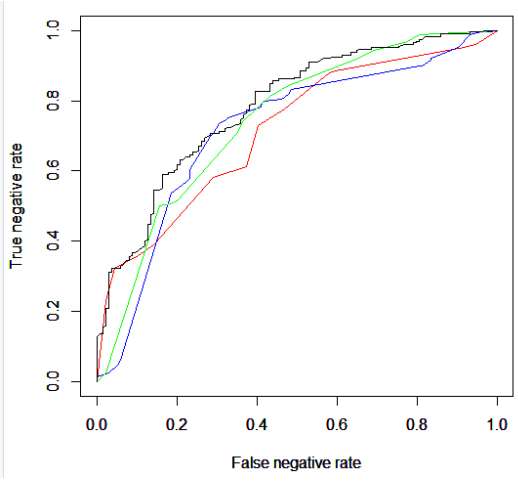
Graphiquement parlant, la courbe rouge (de tree) semble clairement en dessous des deux autres.
Nous pouvons le vérifier avec l'AUC :

```
> # Calcul de la proportion de succès parmi le nombre total d'exemples de test
> sum(df_Tree3[df_Tree3$Classe==df_Tree3$Prediction,"Effectif"])/nrow(ET)
[1] 0.6852368
> #INDICE AUC DU 3eme CLASSIFIEUR
> attr(auc_Tree3, "y.values")
[[1]]
[1] 0.7157711
```

Nous avons effectivement un indice AUC plus bas et un taux de succès très en dessous également des précédents classifieurs.

d) Classifieur « randomForest » (black)

Comme pour les précédents classifieurs, nous comparons les courbes ROC en les superposant.



On peut voir que la courbe noire (randomForest) est au-dessus des autres sur presque tout le graphique. Ce qui indique de bonnes performances.

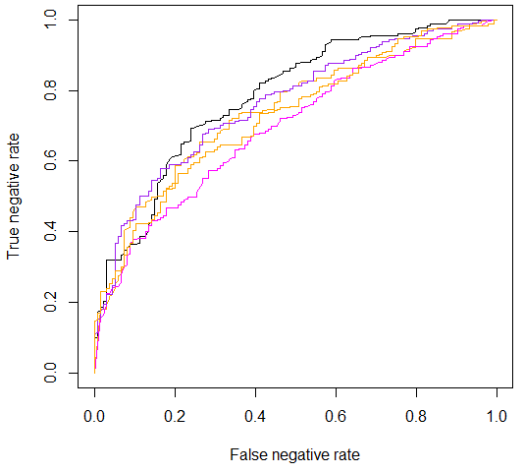
```
> sum(df_Tree4[df_Tree4$Classe==df_Tree3$Prediction,"Effectif"])/nrow(ET)
[1] 0.7214485
> #COURBE ROC ET INDICE AUC
> prob_Tree4 <- predict(Tree4,ET, type="prob")
> roc_pred_Tree4 <- prediction(prob_Tree4[,2],ET$default)
> roc_perf_Tree4 <- performance(roc_pred_Tree4,"tnr","fnr")
> plot(roc_perf_Tree4, col = "black")
> auc_Tree4 <- performance(roc_pred_Tree4, "auc")
> attr(auc_Tree4, "y.values")
[[1]]
[1] 0.7832836
```

On remarque ici qu'il y a un indice AUC bien plus élevé que les précédents classifieurs. Celui-ci reste donc assez intéressant pour nos prédictions. Le taux de succès est lui aussi plus élevé.

e) Classifieur « kknn »

Nous devons créer une fonction pour ce classifieur afin de modifier plus aisément les paramètres (nombre de voisin, distance etc..). Après avoir créer cette fonction nous affichons les courbes ROC et les AUC avec divers paramètres (nous comparons cette courbe à celle de randomForest-la noire- afin de voir si celui-ci est intéressant) :

```
Tree5_1<-Tree5(10,1,TRUE,'orange')
Tree5_2<-Tree5(10,2,TRUE,'magenta')
Tree5_3<-Tree5(20,1,TRUE,'purple')
Tree5_4<-Tree5(20,2,TRUE,'dark green')
Tree5_4<-Tree5(20,2,TRUE,'orange')
```



A vue d'œil, toutes les courbes semblent être moins intéressantes que celle de « **randomForest »** en noire.

```
> Tree5_1<-Tree5(10,1,TRUE,'orange')

      Non Oui
  Non 180  45
  Oui  64  70
AUC =  0.740149253731343> Tree5_2<-Tree5(10,2,TRUE,'magenta')

      Non Oui
  Non 166  59
  Oui  68  66
AUC =  0.695107794361526> Tree5_3<-Tree5(20,1,TRUE,'purple')

      Non Oui
  Non 183  42
  Oui  67  67
AUC =  0.76150912106136> Tree5_4<-Tree5(20,2,TRUE,'dark green')

      Non Oui
  Non 176  49
  Oui  72  62
AUC =  0.725058043117744> Tree5_4<-Tree5(20,2,TRUE,'orange')

      Non Oui
  Non 176  49
  Oui  72  62
AUC =  0.725058043117744
```

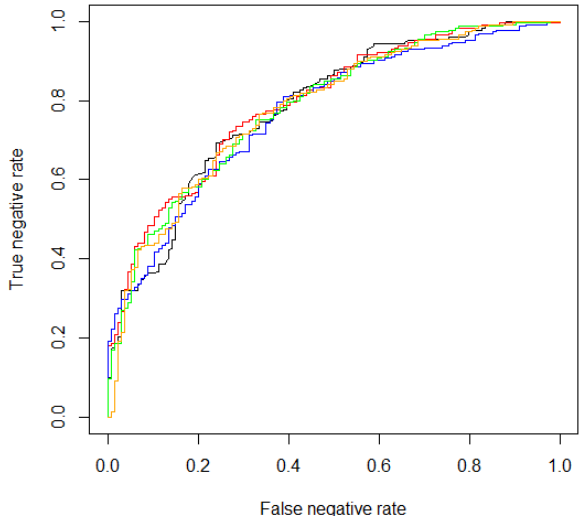
Les indices nous montrent qu'effectivement le classifieur « **randomForest** » est plus optimal pour nos recherches. Utiliser les plus proches voisins ne semble pas être la meilleure méthode pour nos prédictions.

f) Classifieur « svm »

Ce classifieur nécessite aussi de tester des paramétrages différents.
Après avoir créé une fonction pour l'utiliser nous testons :

```
Tree6("linear", TRUE, "red")
Tree6("polynomial", TRUE, "blue")
Tree6("radial", TRUE, "green")
Tree6("sigmoid", TRUE, "orange")
```

Comparons cela à **« randomForest »** qui reste le meilleur jusqu'à maintenant :



On note à l'œil nu que **« randomForest »** pourrait être un peu moins optimal que **« svm ».**

```
> Tree6("linear", TRUE, "red")
     svm_class
      Non Oui
  Non 186  39
  Oui  59  75
AUC =  0.793399668325042> Tree6("polynomial", TRUE, "blue")
     svm_class
      Non Oui
  Non 223   2
  Oui 128   6
AUC =  0.769585406301824> Tree6("radial", TRUE, "green")
     svm_class
      Non Oui
  Non 191  34
  Oui  64  70
AUC =  0.783449419568824> Tree6("sigmoid", TRUE, "orange")
     svm_class
      Non Oui
  Non 185  40
  Oui  60  74
AUC =  0.779701492537313
```

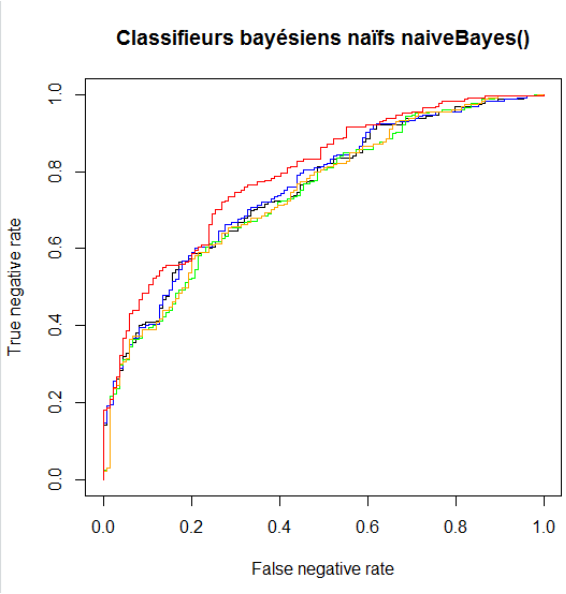
En observant les indices AUC, on note que la fonction utilisant « **svm** » avec le paramètre « **linear** » obtient de meilleur résultat. Son indice AUC est de 0.7934 ce qui est le meilleur que nous ayons obtenu depuis le début.

g) Classifieur « naivebayes »

Ce classifieur nécessite une fonction pour le paramétrer également. Après avoir créé cette fonction, nous testons :

```
# Naive Bayes
Tree7_1<-Tree7(0, FALSE, FALSE, "black")
Tree7_2<-Tree7(20, FALSE, TRUE, "blue")
Tree7_3<-Tree7(0, TRUE, TRUE, "green")
Tree7_4<-Tree7(20, TRUE, TRUE, "orange")
```

D'après les courbes affichées comparées à « **svm** »(rouge), on voit que celle-ci sont moins bien car bien en dessous.



En comparant les indices AUC, on constate que « svm » avec le paramètre « linear » est bien mieux comme classifieur. On ne s'attarde pas sur « naivebayes ».

```
> Tree7_4<-Tree7(20, TRUE, TRUE, "orange")
nb_class
Non Oui
Non 153  72
Oui  46  88
AUC =  0.744212271973465

> Tree7_3<-Tree7(0, TRUE, TRUE, "green")
nb_class
Non Oui
Non 155  70
Oui  50  84
AUC =  0.741724709784411

> Tree7_2<-Tree7(20, FALSE, TRUE, "blue")
nb_class
Non Oui
Non 147  78
Oui  37  97
AUC =  0.758573797678276

> # Naive Bayes
  > Tree7_1<-Tree7(0, FALSE, FALSE, "black")
nb_class
Non Oui
Non 149  76
Oui  41  93
AUC =  0.752968490878939
```

5) Description du classifieur sélectionné

Nous avons donc choisi d'utiliser le classifieur « **svm** » qui obtient de très bon résultat avec le paramètre « **linear** ». Ce n'est pas étonnant car c'est un classifieur qui maximise la marge afin de diminuer les erreurs pour les nouvelles entrées. Celui-ci a utilisé les 11 variables (car nous avons retiré la variable customer), pour les représenter dans un hyperplan à 11 dimensions, ainsi il peut tracer des marges optimales entre les points de même prédictions, afin de mieux prédire de nouveaux points. Sa courbe était clairement plus performante que les autres, ce que nous avons pu voir avec l'indice AUC qui était de 0.79.

6) Résultats sur les données à prédire

On ajoute dans notre fonction « **svm** » :

- print(summary(svm_prob))

Ensuite on la relance avec le paramètre « **linear** » :

```
> Tree6("linear", TRUE, "red")
Non Oui
245 114
     svm_class
      Non Oui
  Non 186  39
  Oui  59  75
      Non                Oui
 Min.   :0.009059   Min.   :0.0009309
 1st Qu.:0.455862   1st Qu.:0.1456291
 Median :0.684419   Median :0.3155808
 Mean   :0.638221   Mean   :0.3617787
 3rd Qu.:0.854371   3rd Qu.:0.5441381
 Max.   :0.999069   Max.   :0.9909405
AUC =  0.793399668325042
```

Cela nous permet d'observer la matrice de confusion, l'indice AUC, mais aussi les minimum, quantile, maximum et médiane des probabilités associées aux deux classes default=oui et default=non.

7) Conclusion

Le meilleur classifieur est donc « svm », avec le paramètre « linear », c'est avec celui-ci que nous aurons le plus de chance de faire de bonnes prédictions.

La difficulté première aura été la création de la fonction permettant d'utiliser correctement ce classifieur.

Par ailleurs, la préparation des données aura été cruciale, car nous voyons que nous avons choisi un classifieur qui crée des hyperplans en se servant de toutes les variables prédictives. Si celles-ci avaient été séparées sans être triées, le classifieur n'aurait pas disposé de toutes les informations pour prédire des choses qu'il n'aura pas vu auparavant (par exemple avec branch). Il en est de même pour les autres modifications comme le retrait de customer etc…

Il ne reste plus qu'à utiliser notre classifieur sur de nouvelles données.