Université Paris Descartes

Projet d'Apprentissage Supervisé

Wafa AYAD - Sina AHMADI

le 16 janvier 2017

Abstract

Le projet actuel représente les travaux des auteurs en tant que compte rendu du projet demandé par M. Lazhar LABIOD. Cela correspond au module d'Apprentissage Supervisé de Master2 Intelligence Artificielle à l'Université Paris Descartes.

Après avoir traité les jeux de données, certains algorithmes d'apprentissage supervisé sont appliquer sur les données. Ce document résume la partie théorique du travail résultant de la partie pratique joint au même dossier rendu.

Table des matières

Introduction

Différentes approches d'apprentissage automatique supervisé existent. Le type de données, leur volume et la disponibilité suffisante de leurs caractéristiques sont des facteurs centraux du choix des algorithmes performants permettant d'avoir un bon rendu d'apprentissage, lequel est nécessaire pour l'étape de prédiction par la suite.

Dans ce rapport, nous allons synthétiser une étude comparative entre plusieurs méthodes d'apprentissage sur des données synthétiques dont les labels sont connus. Ensuite, nous appliquons les modèles de classements adéquats sur un cas des liens d'une banque dans le but d'estimer un score d'appétence à la carte *VISA Premier*.

Ce document est constitué de deux étapes majeurs, la première est consacrée à l'étude comparative entre les algorithmes d'apprentissage supervisé appliqués sur trois jeux de données fournis (*flame.r*, *spiral.r* et *aggregation.r*). Quant à la deuxième partie, et après une étude exploratoire préliminaire de *VisaPremier.txt*, nous discutons l'application des approches que nous jugeons convenables aux données réelles fournisses.

Partie I : Étude sur données synthétiques

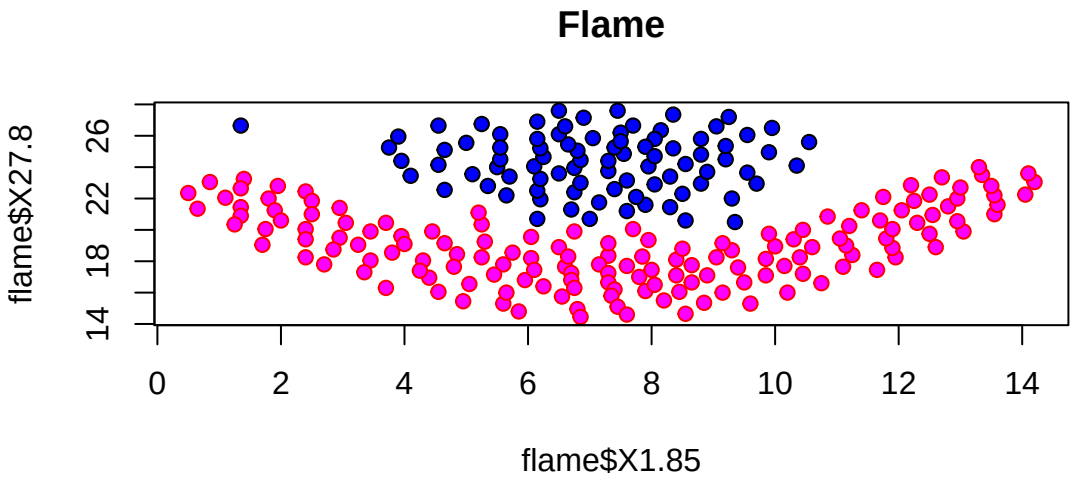
Dans cette partie nous réalisons une étude comparative des différentes approches de classification supervisée vu en cours telles que :

- Logistic Regression
- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- Naive Bayes
- K-nearest Neighbours (KNN)
- Support Vector Machine (SVM)

1 Analyse : *Aggregation.txt*, *Flame.txt* et *Spiral.txt*

1.1 *Flame.txt*

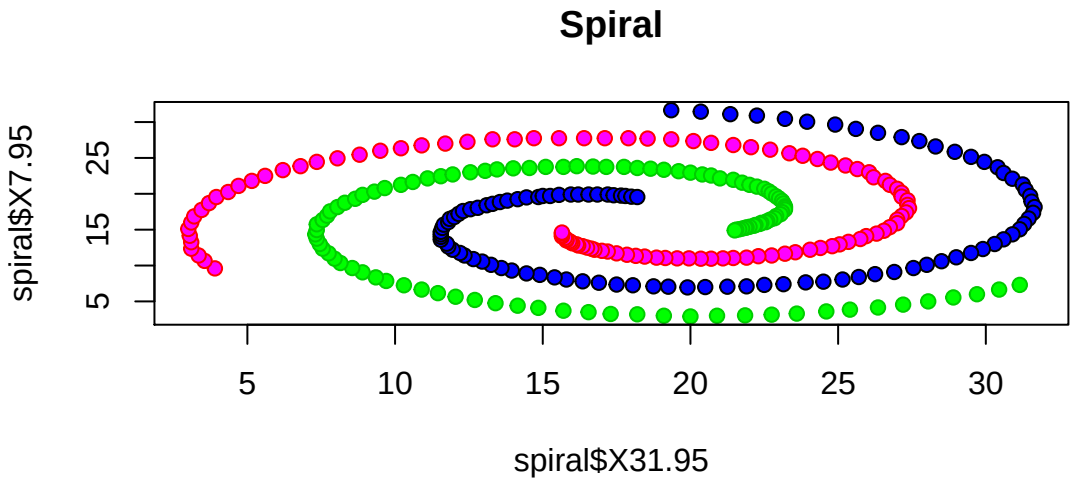
Ces données sont représentées par une matrice de 239 individus décrits par 2 variables de valeurs quantitatives continues. Ces individus sont divisés en 2 classes. Il n'y a pas des valeurs manquantes. Les données n'ont pas besoin de méthodes de prétraitement et peuvent être utilisées directement pour l'apprentissage et la prédiction.



Comme est illustré dans la figure précédente, les données forment 2 classes plus au moins séparées. Une ligne sous forme d'une parabole peut les bien séparer. Les algorithmes linéaires ne pourront pas trouver une bonne marge entre les deux classes dans ce cas-là.

1.2 *Spiral.txt*

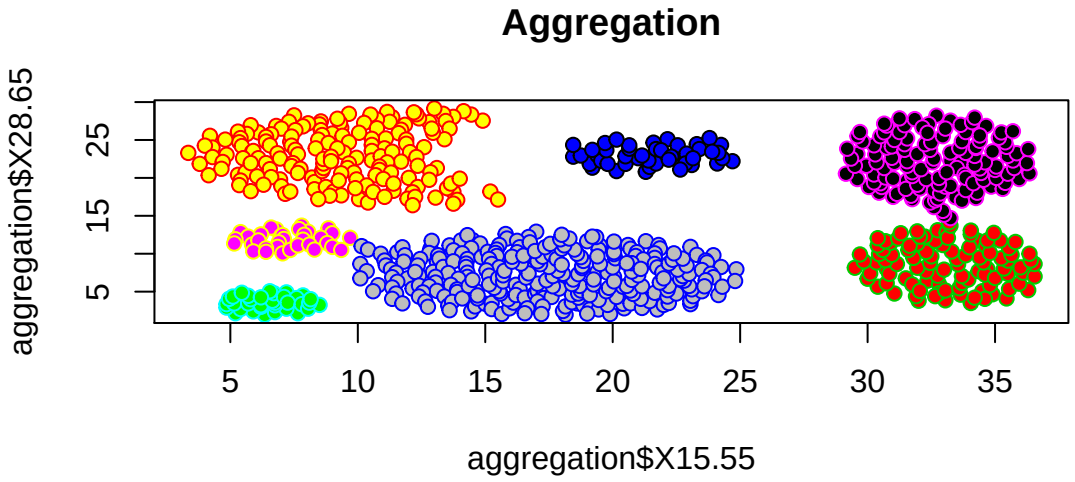
Les 311 individus de cette base sont bivariés, et sont divisés en 3 classes. Aucune des méthodes de prétraitement n'est nécessaire à appliquer sur ces données.



La visualisation des données du fichier *spiral*, comme le montre la figure au dessus, nous permet de distinguer 3 classes spirales bien séparées, les classificateurs linéaires ne pourront pas les distinguer. Cependant, d'autres méthodes non linéaires devraient pouvoir trouver ces 3 classes.

1.3 *Aggregation.txt*

Cette base de données contient 788 individus représentés par 2 variables dont les valeurs sont quantitatives continues. Nous remarquons qu'il n'y a pas de valeurs manquantes. Les intervalles des valeurs pour les deux variables sont comparables, donc nous n'avons pas besoin de prétraitement tels que la mise en échelle ni la normalisation pour entamer la phase d'apprentissage. Les éléments de cette base sont regrouper en 7 classes différentes comme est illustré dans la figure suivante.



Le schéma ci-avant montre une bonne séparation des 7 classes, sauf quelques rares points qui sont à la limite d'une classe et une autre. Il est clair que les séparateurs linéaires (tel que LDA) devraient pouvoir détecter facilement ces 7 classes.

2 Comparaison des modèles de classification supervisée

Dans cette section , nous appliquons les différents modèles de classement des données sur les 3 échantillons fournis. Ensuite, nous synthétisons une étude comparative entre les performances de chaque modèle sur chaque base de données dans trois tableaux. Le code utilisé est disponible dans les fichiers joints *flame.r*, *spiral.r* et *aggregation.r*.

Pour appliquer les différentes approches citées auparavant, nous avons consacré 70% des données de chaque base pour la phase d'apprentissage et 30% pour le test.

2.1 Apprentissage sur *Flame.txt*

Le tableau ci-après résume la précision obtenue par les méthodes citées auparavant sur les données *Flame*. Pour le KNN, nous avons utilisé une boucle (de 1 à 10) pour trouver empiriquement le bon K qui donne par la suite de bonnes performances. Quant au SVM, nous l'avons testé avec plusieurs Kernels à savoir: *linear*, *polynomial*, *radial* et *sigmoid* Les précisions trouvées varient entre celles qui sont bonnes, moyennes et moins bonnes.

Méthode |	Logistic Regression|	LDA |	QDA |	Naive Bayes	|KNN (k=4)|	SVM(Radial Guassien)|
Précision |	0.46	0.36	0.56	0.67	0.92	0.99 |

Pour une bonne classification de la base *Flame*, il est recommandable d'utiliser un SVM avec un noyau radial gaussien car sur un échantillon de 100 individus il ne se trompe que dans le classement d'un seul élément. Le 4NN a aussi un moins d'erreurs de classification. Par contre, LDA, QDA et la régression logistique ne donnent pas de bonnes performances dans l'apprentissage sur les données *Flame*.

2.2 Apprentissage sur *Spiral.txt*

Méthode |	Logistic Regression|	LDA |	QDA |	Naive Bayes	|KNN (k=4)|	SVM(Radial Guassien)|
Précision |	0.33	0.21	0.0	0.0	0.21	0.87

2.3 Apprentissage sur *Aggregation.txt*

Méthode |	Logistic Regression|	LDA |	QDA |	Naive Bayes	|KNN (k=4)|	SVM(Radial Guassien)|
Précision |	0.06	0.36	0.12	0.67	0.92	0.99

3 Synthèse et conclusion

L'exécution des méthodes d'apprentissage citées auparavant a donné des résultats différents d'un modèle à un autre, cela est du au fait que chaque algorithme a ses points forts et points faibles, car le type, la dispersion et le volume des données sont les facteurs sur lesquels les modèles s'appuient.

Partie II: Quelques points importants

1. Étude exploratoire préliminaire

Cette étape joue un rôle primordial pour le reste du projet. Les jeux de données ayant des colonnes et des individus, ne garantissent pas forcément la qualité et la cohérence entre eux. Selon nous c'était exigeant de pré-traiter les jeux de données initiaux pour ce projet.

On observé certains phénomènes en analysant des données. Tout d'abord, les données manquantes se trouvaient dans plusieurs colonnes. Les données manquantes sont les données qui fournissent pas assez données pour qu'elles considèrent pertinentes. Les colonnes ayant des valeurs non cohérentes, répétées et erronées peuvent aussi être supprimées. Le raffinage de bruyance est aussi aussi les variables peu-représentatives peuvent aussi mener d'avoir un jeu de données plus fiable.

Initialement il existait 47 variables et après la phase du pré-traitement, on a pu extraire 34 variables, en plus la variable de la classe.

La normalisation des données était aussi une des tâches initiales qu'on a effectuée sur le jeu des données. Ayant $\sum_{i,j} \frac{x_ij - \overline{x}_j}{\sigma^2}$, on pouvais centrer les données autours de la moyenne des individus et ensuite les normalisant par l'écart-type approprié.

2. Différents algorithmes, différents résultats

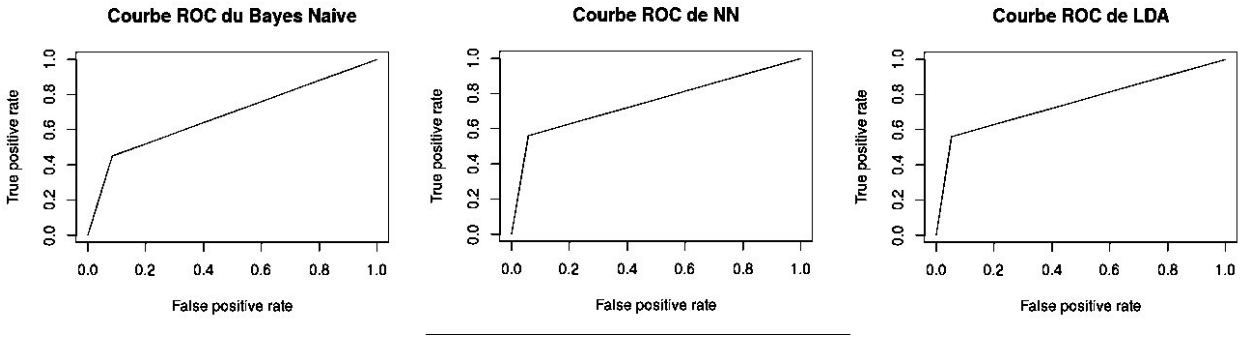
Comme on a vu dans la première partie, chacun des algorithmes peut mener aux résultats qui ne seront pas forcément identiques. Alors que l'algorithme de LDA pour les trois classes, propose une valeur d'estimation

basse mais pertinente, l'algorithme de Naive Bayes ne montre aucune qualité pour la classe Spiral. Pour la même classe on vois que l'algorithme de QDA est aussi égal à 0. Ces valeurs indiquent le lien entre l'essence des données et l'application de chacun des algorithmes à ses propres façons.

Étant les résultats obtenus, on peut bien avouer que les résultats d'algorithme de SVM représente le meilleur. En général, l'algorithme de SVM est un algorithme performant dans le domaine d'analyse de données.

3. Une courbe explicative

Afin de mieux comparer les résultats obtenus par les différents algorithmes, on a essayé de schématiser les résultats de chacun par le diagramme suivant. Le diagramme de ROC (Receiver Operating Characteristic) est un diagramme pour illustrer le rapport sensibilité/spécificité afin d'évaluer la performance de la classification. Les résultats des algorithmes Naive Bayes, LDA et KNN sont représentés dans les diagrammes suivants.



EN comparaison des courbes de ROC, on aperçois que les deux courbes en KNN et LDA partagent les même informations. Plus la courbe est élevée, mieux c'est le modèle. Ce qu'on observe témoigne le fait que les deux algorithmes KNN et LDA ont une qualité meilleure que celle de Naive Bayes.

4 Annexe

L'intégralité des code en R est disponible dans le dossier contenant ce document. Les 3 parties des codes qui viennent ci-dessous sont chacune pour agrégation, flame et spiral. On a essayé de bien mettre les commentaires afin d'avoir une meilleure lisibilité pendant l'exécution.

Le contenu du ficher aggregation.r :

```
1
2  # Redirection vers le bon path
3  setwd("F:/M2-MLDS/Apprentissage supervisé/Projet_app_sup_MLDS16")
4
5  #Chargement des données
6  aggregation <- read.table("Aggregation.txt", header = T,sep="\t")
7
8
9  #Aggregation
10 dim(aggregation)
11 plot(aggregation$X15.55, aggregation$X28.65, col= aggregation$X2,
12     pch=21,bg=c("blue","yellow","red","gray","green", "black", "magenta")
13     [as.numeric(aggregation$X2)], main="Aggregation")
14
15 dim(aggregation)
16 #aggregation[,-3] : Les variables de aggregation
17 #aggregation[,3] : les classes (2 classes) de aggregation
18
19 #Nous eclatons aggregation en 2 échantillons
20 #Apprentissage : 89% aggregation ==>  700 ind pour le training set
21 #Test : 11% aggregation ==> 87 pour test set
22
23 aggregation_train = aggregation[1:87,]
24 aggregation_test = aggregation[88:787,]
```

```
25
26  #Vérification qu'on a bien 61 pour le test et 250 pour l'apprentissage
27  dim(aggregation_train)
28  dim(aggregation_test)
29
30  ##############################################################################################
31  #                                    Apprentissage & test                                    #
32  ##############################################################################################
33
34
35  #*******************************Regression Logistique*********************************#
36  #Apprentissage sur l'ensemble de aggregation_train
37  aggregation.logReg<- glm(aggregation_train[,3] ~.,data=aggregation_train[,-3])
38  #Prédiction
39  aggregation.logReg.pred <- predict(aggregation.logReg,newdata=aggregation_test[,-3],type='response')
40  #Selon le seuil,
41  aggregation.logReg.pred <- ifelse(aggregation.logReg.pred > 0.5,1,0)
42  #Calcul de la précision
43  aggregation.logReg.erreurClass = mean(aggregation.logReg.pred != aggregation_test[,3])
44  aggregation.logReg.accurcy = 1 - aggregation.logReg.erreurClass
45  print(paste("Précision Logistic Regression =",
46              round(aggregation.logReg.accurcy, digits = 2)*100,"%"))
47
48  #La regression logistique donne une précision de 6% :(
49
50  #*************************************LDA*************************************#
51  library(MASS)
52  #Construction de discrimination linéaire
53  aggregation.lda <- lda(aggregation_train[,-3], aggregation_train[,3] )
54  #Prediction selon le modèle linéaire
55  aggregation.lda.pred <- predict(aggregation.lda,aggregation_test[,-3])
56  #Table de confusion
57  table(aggregation.lda.pred$class,aggregation_test[,3])
58
59  #Calcul de la pécision que la lDA donne
60  aggregation.lda.erreurClass <- mean(aggregation.lda.pred$class != aggregation_test[,3])
61  aggregation.lda.accurcy = 1 - aggregation.lda.erreurClass
62  print(paste("Précision LDA =",round(aggregation.lda.accurcy, digits = 2)*100,"%"))
63  #LDA donne une précision de 36% :'(
64
65  #*************************************KNN*************************************#
66  library(class)
67  #Pour trouver de meilleure performance, nous devrions tester l'algorithme KNN pour
68  #différentes valeurs de K, nous ne sauvegardons que le k donnant la meilleure précision.
69  k.optim = 0
70  aggregation.KNN.accuracy = 0
71  for (i in 1:10) {
72    aggregation.KNN = knn(aggregation_train[,-3], aggregation_test[,-3], cl = aggregation_train[,3] , k = i
        )
73    tmp = 1 - ( sum(aggregation.KNN != aggregation_test[,3]) / length(aggregation_test[,3]) )
74    if(tmp > aggregation.KNN.accuracy){
75      aggregation.KNN.accuracy = tmp
76      k.optim = i
77    }
78  }
79
80  print(paste("Précision ",k.optim ,"NN =",round(aggregation.KNN.accuracy, digits = 2)*100,"%"))
81  #La meilleur valeur de K (allant de 1:10 ou méme 1:20) est 4
82  #La précision de 4NN est 92% meilleure que celle de la LDA; :)
83
84
85  #*****************************Classificateur Baysien Naif*****************************#
86
87  library(e1071)
88  # apprentissage sur le training set
89  aggregation.NaiveB = naiveBayes(as.factor(aggregation_train[,3]) ~ ., data = aggregation_train[,-3])
90  #Prediction sur l'échantillon test
91  aggregation.NaiveB.pred = predict(aggregation.NaiveB, aggregation_test[,-3])
92  #Table de confusion
93  table(aggregation.NaiveB.pred,aggregation_test[,3])
94  #Calcul de la précision
95  aggregation.NaiveB.erreurClass <- mean(aggregation.NaiveB.pred != aggregation_test[,3])
96  aggregation.NaiveB.accurcy = 1 - aggregation.NaiveB.erreurClass
97  print(paste("Précision Naive Bayes =",round(aggregation.NaiveB.accurcy, digits = 2)*100,"%"))
98  # Le score obtenu est de 67% mieux que la LDA mais moins bon que KNN :/
99
100
101 #*************************************QDA*************************************#
102 #Construction de discrimination quadratique
103 aggregation.qda <- qda(aggregation_train[,-3], aggregation_train[,3])
104 #Prediction selon le modéle quadratique
```

```
aggregation.qda.pred <- predict(aggregation.qda,aggregation_test[,-3])
#Table de confusion
table(aggregation.qda.pred$class, aggregation_test[,3])
#Calcul de la précision
aggregation.qda.erreurClass <- mean(aggregation.qda.pred$class != aggregation_test[,3])
aggregation.qda.accurcy = 1 - aggregation.qda.erreurClass
print(paste("Précision QDA =",round(aggregation.qda.accurcy, digits = 2)*100,"%"))

#Contrairement é ce qu'on aurait pu penser la QDA ne donne pas de trés bon résultats
#Seulement 12% de précision (mieux que LDA mais moins bonne que KNN et Naive Bayes :/


#######################################SVM#######################################
#Définition des kernls nécissaires au SVM
kernels = c("linear","polynomial","radial","sigmoid")
#La fonction tune éstime les paramétres
aggregation.svmTune = tune(svm, train.x = aggregation_train[,-3], train.y = aggregation_train[,3],
                     validation.x = aggregation_test[,-3], validation.y = aggregation_test[,3],
                     ranges = list(kernel = kernels))

# précision du meilleur modéle
aggregation.svm.accurcy = 1 - aggregation.svmTune$best.performance
print(paste("Précision SVM (",aggregation.svmTune$best.parameters$kernel,") =",
            round(aggregation.svm.accurcy, digits = 2)*100,"%"))
#Avec le kernel gaussien radial, nous obtenons un taux de précision supérieur é 99% :D
```

Le contenu du fichier spiral.r :

```
# Redirection vers le bon path
setwd("F:/M2-MLDS/Apprentissage supervisé/Projet_app_sup_MLDS16")

#Chargement des données
spiral <- read.table("spiral.txt",header = T, sep = "\t", fill = T)

#spiral
dim(spiral)
plot(spiral$X31.95, spiral$X7.95, col= spiral$X3, pch=21,bg=c("blue", "magenta", "green")[as.numeric(
  spiral$X3)], main="spiral")

#spiral[,-3] : Les variables de spiral
#spiral[,3] : les classes (2 classes) de spiral

#Nous eclatons spiral en 2 échantillons (é 1/3 de la classe 1, 1/3 de la classe 2 et 1/3 de la classe 3)
#Apprentissage : 80% spiral ==>  250 ind pour le training set
#Test : 20% spiral ==> 61 pour test set

spiral_train = rbind(spiral[21:105,],spiral[126:206,], spiral[228:311, ])
spiral_test = rbind(spiral[1:20,], spiral[106:125,], spiral[207:227, ])

#Vérification qu'on a bien 61 pour le test et 250 pour l'apprentissage
dim(spiral_train)
dim(spiral_test)

#La précision, et la spécificité semblent étre des bonnes mesures pour ésitmer les
#méthodes sur ce jeu de données
#Le nombre d'individu par classe est assez similaire

##############################################################################################
#                                    Apprentissage & test                                    #
##############################################################################################


#################################Regression Logistique#################################
#Apprentissage sur l'ensemble de spiral_train
spiral.logReg<- glm(spiral_train[,3] ~.,data=spiral_train[,-3])
#Prédiction
spiral.logReg.pred <- predict(spiral.logReg,newdata=spiral_test[,-3],type='response')
#Selon le seuil,
spiral.logReg.pred <- ifelse(spiral.logReg.pred > 0.5,1,0)
#Calcul de la précision
spiral.logReg.erreurClass = mean(spiral.logReg.pred != spiral_test[,3])
spiral.logReg.accurcy = 1 - spiral.logReg.erreurClass
print(paste("Précision Logistic Regression =",
            round(spiral.logReg.accurcy, digits = 2)*100,"%"))

#La regression logistique donne une précision de 33% :(

#######################################LDA#######################################
library(MASS)
#Construction de discrimination linéaire
```

```
spiral.lda <- lda(spiral_train[,-3], spiral_train[,3] )
#Prediction selon le modéle linéaire
spiral.lda.pred <- predict(spiral.lda,spiral_test[,-3])
#Table de confusion
table(spiral.lda.pred$class,spiral_test[,3])

#Calcul de la pécision que la lDA donne
spiral.lda.erreurClass <- mean(spiral.lda.pred$class != spiral_test[,3])
spiral.lda.accurcy = 1 - spiral.lda.erreurClass
print(paste("Précision LDA =",round(spiral.lda.accurcy, digits = 2)*100,"%"))
#LDA donne une précision de 0% :'(, si t'as dis je calcule pas éa aurait été mieux :/

#########################################KNN#########################################
library(class)
#Pour trouver de meilleure performance, nous devrions tester l'algorithme KNN pour
#différentes valeurs de K, nous ne sauvegardons que le k donnant la meilleure précision.
k.optim = 0
spiral.KNN.accuracy = 0
for (i in 1:10) {
  spiral.KNN = knn(spiral_train[,-3], spiral_test[,-3], cl = spiral_train[,3] , k = i)
  tmp = 1 - ( sum(spiral.KNN != spiral_test[,3]) / length(spiral_test[,3]) )
  if(tmp > spiral.KNN.accuracy){
    spiral.KNN.accuracy = tmp
    k.optim = i
  }
}

print(paste("Précision ",k.optim ,"NN =",round(spiral.KNN.accuracy, digits = 2)*100,"%"))
#La meilleur valeur de K (allant de 1:10 ou méme 1:20) est 1, ce qui est logique car
#les spirales sont loins l'une des autres !
#La précision de 1NN est 21% , Deéue :( !


############################Classificateur Baysien Naif############################

library(e1071)
# apprentissage sur le training set
spiral.NaiveB = naiveBayes(as.factor(spiral_train[,3]) ~ ., data = spiral_train[,-3])
#Prediction sur l'échantillon test
spiral.NaiveB.pred = predict(spiral.NaiveB, spiral_test[,-3])
#Table de confusion
table(spiral.NaiveB.pred,spiral_test[,3])
#Calcul de la précision
spiral.NaiveB.erreurClass <- mean(spiral.NaiveB.pred != spiral_test[,3])
spiral.NaiveB.accurcy = 1 - spiral.NaiveB.erreurClass
print(paste("Précision Naive Bayes =",round(spiral.NaiveB.accurcy, digits = 2)*100,"%"))
# Le score obtenu est de 0% (au moins toi t'es honéte -_-)


#########################################QDA#########################################
#Construction de discrimination quadratique
spiral.qda <- qda(spiral_train[,-3], spiral_train[,3])
#Prediction selon le modéle quadratique
spiral.qda.pred <- predict(spiral.qda,spiral_test[,-3])
#Table de confusion
table(spiral.qda.pred$class, spiral_test[,3])
#Calcul de la précision
spiral.qda.erreurClass <- mean(spiral.qda.pred$class != spiral_test[,3])
spiral.qda.accurcy = 1 - spiral.qda.erreurClass
print(paste("Précision QDA =",round(spiral.qda.accurcy, digits = 2)*100,"%"))

# 0% :/


#########################################SVM#########################################
#Définition des kernls nécissaires au SVM
kernels = c("linear","polynomial","radial","sigmoid")
#La fonction tune éstime les paramétres
spiral.svmTune = tune(svm, train.x = spiral_train[,-3], train.y = spiral_train[,3],
                      validation.x = spiral_test[,-3], validation.y = spiral_test[,3],
                      ranges = list(kernel = kernels))

# précision du meilleur modéle
spiral.svm.accurcy = 1 - spiral.svmTune$best.performance
print(paste("Précision SVM (",spiral.svmTune$best.parameters$kernel,") =",
            round(spiral.svm.accurcy, digits = 2)*100,"%"))
#Avec le kernel gaussien radial, nous obtenons un taux de précision supérieur é 87% :D
```

Le contenu du fichier flame.r :

```
# Redirection vers le bon path
setwd("F:/M2-MLDS/Apprentissage supervisé/Projet_app_sup_MLDS16")

#Chargement des données
flame <- read.table("flame.txt",header = T, sep = "\t", fill = T)

#Dimension de flame
dim(flame)
#affichage des individus de flame en fonction des 2 variables et en les colorie
#en fonction de leur classe
plot(flame$X1.85, flame$X27.8, col=flame$X1, pch=21,bg=c("blue", "magenta")
     [as.numeric(flame$X1)], main="Flame")
#Il est clair que les 2 classes sont séparées par une courbe parabolique
#(de type y=x0+x1^2)


#flame[,-3] : Les variables de flame
#Flame[,3] : les classes (2 classes) de flame

#Nous eclatons flame en 2 échantillons (à 1/2 de la classe 1 et 1/2 de la classe 2)
#Apprentissage : 84% flame ==>  200 ind pour le training set
#Test : 16% flame ==> 39 pour test set

flame_train = rbind(flame[1:133,],flame[173:240,])
flame_test = flame[134:172,]

#Vérification qu'on a bien 39 pour le test et 200 pour l'apprentissage
dim(flame_train)
dim(flame_test)

#La précision, et la spécificité semblent étre des bonnes mesures pour ésitmer les
#méthodes sur ce jeu de données
#Le nombre d'individu par classe est assez similaire

##############################################################################################
#                                    Apprentissage & test                                    #
##############################################################################################


#*******************************Regression Logistique*********************************#
#Apprentissage sur l'ensemble de flame_train
flame.logReg<- glm(flame_train[,3] ~.,data=flame_train[,-3])
#Prédiction
flame.logReg.pred <- predict(flame.logReg,newdata=flame_test[,-3],type='response')
#Selon le seuil,
flame.logReg.pred <- ifelse(flame.logReg.pred > 0.5,1,0)
#Calcul de la précision
flame.logReg.erreurClass = mean(flame.logReg.pred != flame_test[,3])
flame.logReg.accurcy = 1 - flame.logReg.erreurClass
print(paste("Précision Logistic Regression =",
            round(flame.logReg.accurcy, digits = 2)*100,"%"))

#La regression logistique donne une précision de 46% :(

#*************************************LDA*************************************#
library(MASS)
#Construction de discrimination linéaire
flame.lda <- lda(flame_train[,-3], flame_train[,3] )
#Prediction selon le modéle linéaire
flame.lda.pred <- predict(flame.lda,flame_test[,-3])
#Table de confusion
table(flame.lda.pred$class,flame_test[,3])

#Calcul de la pécision que la lDA donne
flame.lda.erreurClass <- mean(flame.lda.pred$class != flame_test[,3])
flame.lda.accurcy = 1 - flame.lda.erreurClass
print(paste("Précision LDA =",round(flame.lda.accurcy, digits = 2)*100,"%"))
#LDA donne une précision de 36% :'(

#*************************************KNN*************************************#
library(class)
#Pour trouver de meilleure performance, nous devrions tester l'algorithme KNN pour
#différentes valeurs de K, nous ne sauvegardons que le k donnant la meilleure précision.
k.optim = 0
flame.KNN.accuracy = 0
for (i in 1:10) {
  flame.KNN = knn(flame_train[,-3], flame_test[,-3], cl = flame_train[,3] , k = i)
  tmp = 1 - ( sum(flame.KNN != flame_test[,3]) / length(flame_test[,3]) )
  if(tmp > flame.KNN.accuracy){
    flame.KNN.accuracy = tmp
    k.optim = i
```

```
  }
}

print(paste("Précision ",k.optim ,"NN =",round(flame.KNN.accuracy, digits = 2)*100,"%"))
#La meilleur valeur de K (allant de 1:10 ou méme 1:20) est 4
#La précision de 4NN est 92% meilleure que celle de la LDA; :)


#****************************Classificateur Baysien if*******************************#

library(e1071)
# apprentissage sur le training set
flame.NaiveB = naiveBayes(as.factor(flame_train[,3]) ~ ., data = flame_train[,-3])
#Prediction sur l'échantillon test
flame.NaiveB.pred = predict(flame.NaiveB, flame_test[,-3])
#Table de confusion
table(flame.NaiveB.pred,flame_test[,3])
#Calcul de la précision
flame.NaiveB.erreurClass <- mean(flame.NaiveB.pred != flame_test[,3])
flame.NaiveB.accurcy = 1 - flame.NaiveB.erreurClass
print(paste("Précision Naive Bayes =",round(flame.NaiveB.accurcy, digits = 2)*100,"%"))
# Le score obtenu est de 67% mieux que la LDA mais moins bon que KNN :/


#*************************************QDA*****************************************#
#Construction de discrimination quadratique
flame.qda <- qda(flame_train[,-3], flame_train[,3])
#Prediction selon le modéle quadratique
flame.qda.pred <- predict(flame.qda,flame_test[,-3])
#Table de confusion
table(flame.qda.pred$class, flame_test[,3])
#Calcul de la précision
flame.qda.erreurClass <- mean(flame.qda.pred$class != flame_test[,3])
flame.qda.accurcy = 1 - flame.qda.erreurClass
print(paste("Précision QDA =",round(flame.qda.accurcy, digits = 2)*100,"%"))

#Contrairement à ce qu'on aurait pu penser la QDA ne donne pas de trés bon résultats
#Seulement 56% de précision (mieux que LDA mais moins bonne que KNN et Naive Bayes :/


#*************************************SVM*****************************************#
#Définition des kernls nécissaires au SVM
kernels = c("linear","polynomial","radial","sigmoid")
#La fonction tune éstime les paramétres
flame.svmTune = tune(svm, train.x = flame_train[,-3], train.y = flame_train[,3],
                     validation.x = flame_test[,-3], validation.y = flame_test[,3],
                     ranges = list(kernel = kernels))

# précision du meilleur modéle
flame.svm.accurcy = 1 - flame.svmTune$best.performance
print(paste("Précision SVM (",flame.svmTune$best.parameters$kernel,") =",
            round(flame.svm.accurcy, digits = 2)*100,"%"))
#Avec le kernel gaussien radial, nous obtenons un taux de précision supérieur é 99% :D
```

References

[1] Keil, P., Belmaker, J., Wilson, A. M., Unitt, P., & Jetz, W. (2013). Downscaling of species distribution models: a hierarchical approach. *Methods in Ecology and Evolution*, 4(1), 82-94.

[2] Team, R. Core. "R: A language and environment for statistical computing." (2013): 409.

[3] Halpern, B. S., Regan, H. M., Possingham, H. P., & McCarthy, M. A. (2006). Accounting for uncertainty in marine reserve design. *Ecology Letters*, 9(1), 2-11.

.