Projet méthodes avancées en apprentissage supervisé et non-supervisé

Romain Dudoit, Franck Doronzo, Marie Vachet

7 janvier 2022

Table des matières

1 Introduction

Dans le cadre de l'UE Data Mining - Apprentissage statistique, nous avons travaillé sur une étude de cas visant à résoudre un problème de classification ; l'objectif étant pour nous de mettre en application des méthodes d'apprentissage vues en cours et de parfaire notre maîtrise du langage R.
Dans un premier temps, nous présenterons le jeu de données que nous avons choisi ainsi que la problématique sur laquelle nous avons travaillé. Dans les parties suivantes, vous retrouverez la démarche effectuée pour les différents protocoles expérimentaux. Les codes et l'ensemble de nos travaux sont disponibles sur GitHub.

2 Jeu de données et problématique

2.1 Présentation des données

Le jeu de données choisi est disponible sur le site UCI Machine Learning Repository : Dry Bean Dataset. Il porte sur les caractéristiques de grains d'haricots secs. Il comporte 13611 observations pour 7 espèces différentes. Les données ont été obtenues après des étapes de segmentation et d'extraction de caractéristiques sur des images prises par un système de vision par ordinateur utilisant une caméra haute définition.

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	Sh
1	28395	610.291	208.1781	173.8887	1.197191	0.5498122	28715	190.1411	0.7639225	0.9888560	0.9580271	0.9133578	
2	28734	638.018	200.5248	182.7344	1.097356	0.4117853	29172	191.2728	0.7839681	0.9849856	0.8870336	0.9538608	
3	29380	624.110	212.8261	175.9311	1.209713	0.5627273	29690	193.4109	0.7781132	0.9895588	0.9478495	0.9087742	
4	30008	645.884	210.5580	182.5165	1.153638	0.4986160	30724	195.4671	0.7826813	0.9766957	0.9039364	0.9283288	
5	30140	620.134	201.8479	190.2793	1.060798	0.3336797	30417	195.8965	0.7730980	0.9908933	0.9848771	0.9705155	
6	30279	634.927	212.5606	181.5102	1.171067	0.5204007	30600	196.3477	0.7756885	0.9895098	0.9438518	0.9237260	
7	30477	670.033	211.0502	184.0391	1.146768	0.4894779	30970	196.9886	0.7624015	0.9840814	0.8530799	0.9333736	
8	30519	629.727	212.9968	182.7372	1.165591	0.5137596	30847	197.1243	0.7706818	0.9893669	0.9671092	0.9254804	
9	30685	635.681	213.5341	183.1571	1.165852	0.5140809	31044	197.6597	0.7715615	0.9884358	0.9542398	0.9256585	
10	30834	631.934	217.2278	180.8975	1.200834	0.5536422	31120	198.1390	0.7836828	0.9908098	0.9702782	0.9121254	
11	30917	640.765	213.5601	184.4399	1.157885	0.5041024	31280	198.4055	0.7708053	0.9883951	0.9462582	0.9290383	
12	31091	638.558	210.4863	188.3268	1.117665	0.4466219	31458	198.9630	0.7863773	0.9883337	0.9581728	0.9452543	
13	31107	640.594	214.6485	184.9693	1.160455	0.5073659	31423	199.0142	0.7610461	0.9899437	0.9525818	0.9271632	
14	31158	642.626	216.4848	183.6443	1.178827	0.5295143	31492	199.1773	0.7987592	0.9893941	0.9481190	0.9200520	
15	31158	641.105	212.0670	187.1930	1.132879	0.4699242	31474	199.1773	0.7813135	0.9899600	0.9526231	0.9392189	
16	31178	636.888	212.9759	186.5621	1.141582	0.4823522	31520	199.2412	0.7641105	0.9891497	0.9658996	0.9355105	

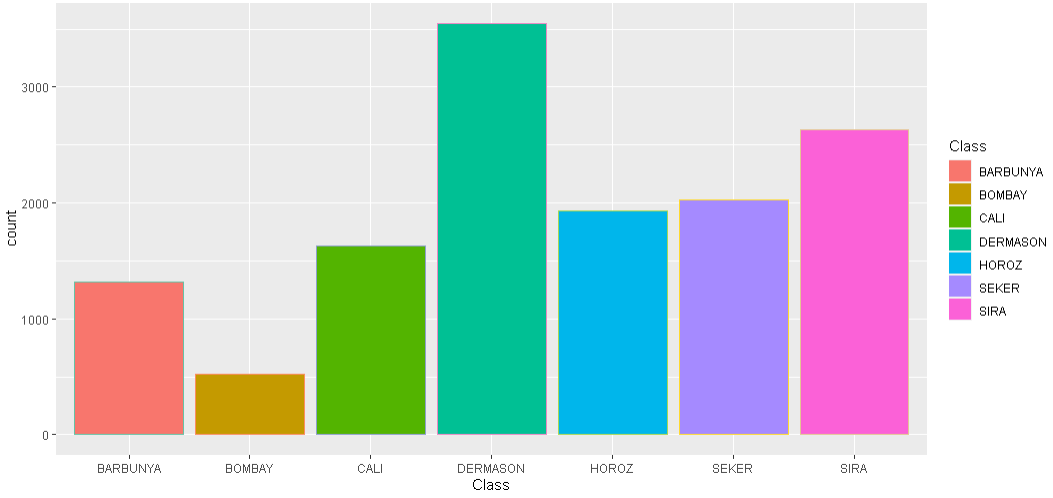
Les variables explicatives sont au nombre de 16 et sont relatives aux caractéristiques des grains :

1.) Area (A) : La surface d'une zone de haricot et le nombre de pixels à l'intérieur de ses limites.
2.) Perimeter (P) : La circonférence d'un haricot est définie comme la longueur de sa bordure.
3.) Major axis length (L) : La distance entre les extrémités de la ligne la plus longue qui peut être tracée à partir d'un haricot.
4.) Minor axis length (l) : La plus longue ligne que l'on peut tracer à partir d'un haricot en étant perpendiculaire à l'axe principal.
5.) Aspect ratio (K) : Définit la relation entre L et l.
6.) Eccentricity (Ec) : Excentricité de l'ellipse ayant les mêmes moments que la région.
7.) Convex area (C) : Nombre de pixels dans le plus petit polygone convexe qui peut contenir la surface d'une graine de haricot.

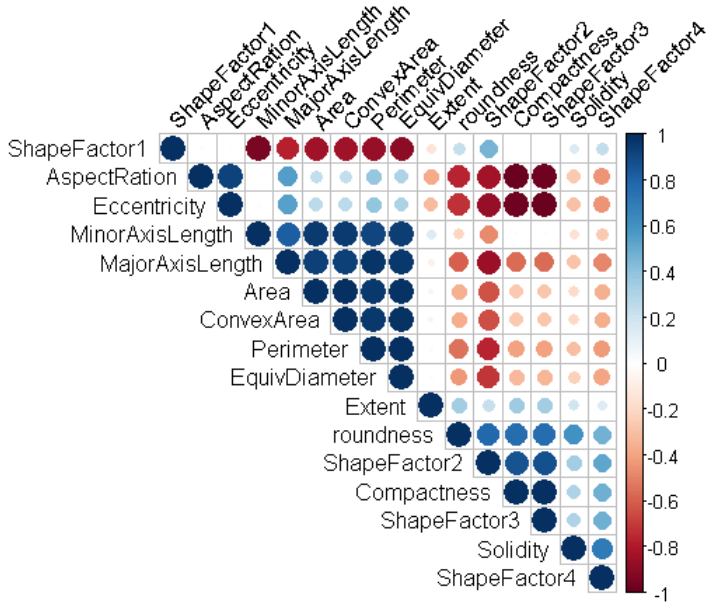
8.) Equivalent diameter (Ed) : Le diamètre d'un cercle ayant la même surface que celle d'une graine de haricot.
9.) Extent (Ex) : Le rapport entre les pixels de la boîte englobante et la surface du haricot.
10.) Solidity (S) : Le rapport entre les pixels de la coquille convexe et ceux que l'on trouve dans les haricots.
11.) Roundness (R) : Calculée à l'aide de la formule suivante : $\frac{4\pi A}{P^2}$
12.) Compactness (CO) : Mesure la rondeur d'un objet : $\frac{Ed}{L}$
13.) ShapeFactor1 (SF1) = $\frac{L}{A}$
14.) ShapeFactor2 (SF2) = $\frac{l}{A}$
15.) ShapeFactor3 (SF3) = $\frac{A}{\frac{L}{2}\cdot\frac{L}{2}\cdot\pi}$
16.) ShapeFactor4 (SF4) = $\frac{A}{\frac{L}{2}\cdot\frac{l}{2}\cdot\pi}$
17.) Class (Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira) : La variable cible que l'on souhaite prédire.

2.2 Statistiques descriptives

Concernant la répartition de la variable cible, on constate que la modalité "DERMASON" est la plus représentée au sein du jeu de données, à l'inverse la modalité la moins représentée est "BOMBAY".



Voici les corrélations entre les différentes variables explicatives :



On peut voir graphiquement que beaucoup de variables sont corrélées.

2.3 Problématique

Notre travail porte sur la problématique suivante : trouver, parmi les modèles imposés celui qui permet de réaliser les meilleures prédictions de la variable cible. Nous nous situons donc dans le cadre d'une classification multiclasse (non binaire) puisque la variable Class possède 7 modalités. Les modèles qui nous étaient imposés étant tous supervisés, nous utiliserons la variable cible dans le processus d'apprentissage.

3 Méthodes utilisées et librairies

Nous allons appliquer comme demandé les méthodes suivantes :

- Modèle linéaire pénalisé par une fonction de régularisation elasticnet
- Machine à vecteur de support (SVM)
- Réseau de neurones avec une couche

Nous adapterons évidemment nos modèles pour la classification.

3.1 Modèle linéaire pénalisé par une fonction de régularisation elasticnet

Comme nous sommes dans un problème de classification multinomiale, le modèle linéaire utilisé sera une régression logistique multinomiale sur laquelle on introduira une pénalité par une fonction de régularisation elasticnet.

L'objectif de la régularisation est d'éviter le surapprentissage en acceptant une légère augmentation du biais durant l'apprentissage pour obtenir une réduction de la variance avec les données de test. La classification

pénalisée permet ainsi d'imposer un coût supplémentaire au modèle pour les erreurs de classification commises sur les classes minoritaires.

La régularisation elasticnet combine les avantages des régularisations ridge (L2) et lasso (L1).
Les paramètres à optimiser sont :

- λ : coefficient de pénalité qui permet de donner un poids plus ou moins important à la pénalité que l'on veut appliquer.
- α : paramètre compris entre 0 et 1 qui permet de combiner les normes l1 et l2 pour obtenir une solution moins parcimonieuse que lasso, mais plus stable. Si $\alpha = 0$ alors la régularisation elasticnet est équivalente à une régularisation ridge. Si α est égale à 1, alors la régularisation elasticnet sera équivalente à une régularisation lasso.

Pour implémenter la régression logistique nous avons utilisé le package glmnet.

3.2 SVM

Le but des SVM est de trouver un hyperplan optimal (marge optimale) pour séparer nos classes, celui-ci se base sur la méthode des classifieurs linéaires. Tout comme les modèles linéaires celui-ci va chercher la frontière optimale de séparation des classes. Pour atteindre ce but d'optimisation des marges il y a deux possibilités :

-on définit un hyperplan comme solution du problème d'optimisation sous contraintes dont la fonction objectif s'exprime à l'aide de produits scalaires entre vecteurs et dans lequel le nombre de contraintes "actives" ou vecteurs supports contrôle la complexité du modèle.

-Le passage à la recherche de surfaces séparatrices non linéaires est obtenu par l'introduction d'une fonction noyau (kernel) dans le produit scalaire induisant implicitement une transformation non linéaire des données vers un espace intermédiaire (feature space) de plus grande dimension. D'où l'appellation couramment rencontrée de machine à noyau ou kernel machine. Sur le plan théorique, la fonction noyau définit un espace hilbertien, dit auto-reproduisant et isométrique par la transformation non linéaire de l'espace initial et dans lequel est résolu le problème linéaire.

Pour implémenter les SVM nous avons utilisé la librairie e1071. De plus cette libraire fournit les noyaux les plus courants, notamment linéaire, polynomial, RBF et sigmoïde.Elle offre une puissance de calcul pour les valeurs de décision et de probabilité pour les prédictions.

3.3 Réseau de neurones avec couche cachée

Les réseaux de neurones artificiels s'inspirent du fonctionnement des neurones du cerveau. Un réseau de neurones est construit par l'association d'un ou plusieurs perceptrons. Un perceptron reçoit en entrée des données provenant d'autres perceptrons (ou des variables) et retourne en sortie un signal qui peut être adressé à d'autres perceptrons ou être les prédictions du modèle. Les réseaux de neurones se distinguent par

des couches (c'est-à-dire des perceptrons) qui représentent leur architecture, par le nombre de neurones dans chaque couche, par le type de neurones (sa fonction d'activation) et par les poids qui leur sont associés. Nous utilisons la librairie *nnet* pour créer un perceptron avec une couche cachée. Nous chercherons à optimiser le nombre de neurones ainsi qu'un paramètre de régularisation (weight decay) afin d'éviter le surapprentissage.

4 Protocole expérimental

Dans un premier temps, nous avons divisé notre jeu de données en deux sous ensembles : un premier jeu d'apprentissage (70%) et un second de test (30%). Étant donné qu'un bon nombre de variables sont corrélées entre elles, nous avons décidé de garder l'ensemble du jeu de données. On applique par la suite une standardisation des données sur l'ensemble des variables explicatives.

Pour la création de nos modèles, on définie les hyperparamètres que l'on souhaite modifier et on effectue un gridsearch avec une cross-validation afin de pouvoir récupérer les meilleurs hyperparamètres qui maximise la mesure de notre choix, en l'occurence le taux de reconnaissance, c'est-à-dire "l'accuracy" de notre modèle. On effectue ensuite les prédictions sur l'échantillon de test et on sort les métriques. Afin de comparer leurs performances, on peut effectuer un 'plot' sur la cross-validation résultant du gridsearch et comparer l'accuracy et autres métriques. (Cette comparaison sera présentée dans la section suivante.)

La standardisation et la cross-validation de certains de nos modèles on été effectuée en utilisant le package *caret* qui est un package contenant des fonctions permettant de rationaliser le processus d'apprentissage du modèle pour les problèmes complexes de régression et de classification.

4.1 Présentation et analyse, des résultats des modèles pour chaque méthode

4.1.1 Modèle linéaire pénalisé par une fonction de régularisation elastinet

À l'aide du package *caret* nous avons effectué une validation croisée avec 5 folds (en spécifiant la méthode *glmnet* pour la classification multinomiale) pour différentes combinaisons des paramètres α et λ. Nous avons fait varier α entre 0 et 1 avec un pas de 0.1 et nous avons généré 20 valeurs différentes de λ comprises entre 0.0001 et 0.1. Pour le paramètre nous n'avons pas testé avec plus de valeurs en raison des temps de calculs et de précédents essais nous on montré qu'il n'est pas intéressant d'aller au-delà de 0.1
À l'issue de cette validation croisée, le meilleur modèle qui à maximisé le taux de reconnaissance à été celui ayant pour paramètre $\lambda = 0.0001$ et $\alpha = 1$.

Nous avons ensuite utilisé ces paramètres pour entrainer notre modèle avec le package *glmnet* qui est un package qui s'adapte à des modèles linéaires généralisés et similaires via un maximum de vraisemblance pénalisé. Nous avons obtenu sur l'échantillon de test un score d'accuracy de 0.92. Voici l'ensemble des statistiques obtenues :

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      358      0   11        1     1     7    0
##   BOMBAY          0    156    0        0     0     0    0
##   CALI           23      0  469        0    16     0    2
##   DERMASON        0      0    0      973     7    12   86
##   HOROZ           0      0    7        1   545     0   13
##   SEKER           6      0    2       20     0   573   16
##   SIRA            9      0    0       68     9    16  673
##
## Overall Statistics
##
##                Accuracy : 0.9184
##                  95% CI : (0.9096, 0.9266)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.9013
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BARBUNYA Class: BOMBAY Class: CALI Class: DERMASON
## Sensitivity                  0.90404       1.00000      0.9591          0.9153
## Specificity                  0.99457       1.00000      0.9886          0.9652
## Pos Pred Value               0.94709       1.00000      0.9196          0.9026
## Neg Pred Value               0.98974       1.00000      0.9944          0.9700
## Prevalence                   0.09706       0.03824      0.1199          0.2605
## Detection Rate               0.08775       0.03824      0.1150          0.2385
## Detection Prevalence         0.09265       0.03824      0.1250          0.2642
## Balanced Accuracy            0.94931       1.00000      0.9738          0.9403
##                      Class: HOROZ Class: SEKER Class: SIRA
## Sensitivity                0.9429       0.9424      0.8519
## Specificity                0.9940       0.9873      0.9690
## Pos Pred Value             0.9629       0.9287      0.8684
## Neg Pred Value             0.9906       0.9899      0.9646
## Prevalence                 0.1417       0.1490      0.1936
## Detection Rate             0.1336       0.1404      0.1650
## Detection Prevalence       0.1387       0.1512      0.1900
## Balanced Accuracy          0.9685       0.9649      0.9104
```

4.1.2 SVM

Afin d'évaluer le modèle du SVM la librairie *e1071* permet d'effectuer une validation croisée afin de renvoyer les meilleurs hyperparamètres. L'hyperparamètre C (cout) est responsable de la taille de la marge du MVC, les points situés à l'intérieur de cette marge ne sont classés dans aucune des deux catégories. Plus la valeur de C est faible, plus la marge est importante, ici pour la validation croisée nous avons décidé de faire varier C de 0.1 à 10 avec un pas de 0.1. Le kernel lui permet de rendre linéairement séparable un problème qui ne le serai pas dans l'espace de base, ici nous testerons tous les kernel disponible ai sein de la librairie *e1071*.

Ainsi on peut se rendre compte que le modèle ayant les meilleures performances est celui dont les hyperparamètres sont : Kernel : Radial Cout : 7

```
tuned = tune(svm,train.x=as.matrix(XTrain_scaled),
             train.y=yTrain, data = data,
             scale=F, type = "C-classification",
             ranges = list(cost=seq(0.1, 10, 0.1),kernel= c("linear","radial","sigmoid","polynomial")),
             tunecontrol=tune.control(cross=10))
tuned$performances
```

cost <dbl>	kernel <fctr>	error <dbl>	dispersion <dbl>
6.1	radial	0.06945660	0.007536135
6.2	radial	0.06914180	0.007817045
6.3	radial	0.06914180	0.007817045
6.4	radial	0.06903687	0.007600287
6.5	radial	0.06903687	0.007600287
6.6	radial	0.06872208	0.007607148
6.7	radial	0.06893194	0.007457980
6.8	radial	0.06872208	0.007227795
6.9	radial	0.06872208	0.007227795
7.0	radial	0.06851221	0.007291568

161-170 of 400 rows Previous 1 … 15

FIGURE 1 – GridSearch SVM

4.1.3 Réseaux de neurones avec une couche cachée

Pour évaluer au mieux notre modèle, nous avons choisi d'effectuer une validation croisée de type k-fold avec k=5. De plus, nous avons décidé de chercher quel est le meilleur nombre de neurones à utiliser dans la couche cachée ainsi que le coefficient de régularisation (weight decay ou dégradation des pondérations). Nous avons testé les valeurs suivantes pour decay : 0.5, 0.3, 0.1, 0.001 et ces valeurs pour le nombre de neurones : 10, 12, 14, 16.
Voici le tableau des résultats :

decay	size	AUC	Accuracy	Mean$_F$1
0.001	10	0.9950756	0.9281300	0.9392268
0.001	12	0.9951638	0.9282353	0.9392946
0.001	14	0.9949013	0.9278150	0.9384273
0.001	16	0.9944710	0.9266609	0.9371787
0.100	10	0.9957846	0.9277103	0.9388759
0.100	12	0.9958467	0.9305436	0.9423791
0.100	14	0.9959519	0.9294939	0.9406108
0.100	16	0.9959439	0.9311733	0.9424980
0.300	10	0.9955931	0.9294943	0.9401507
0.300	12	0.9957747	0.9305433	0.9420282
0.300	14	0.9958168	0.9306486	0.9417651
0.300	16	0.9959176	0.9300190	0.9411577
0.500	10	0.9955344	0.9286551	0.9394638
0.500	12	0.9956761	0.9295991	0.9400968
0.500	14	0.9957118	0.9298090	0.9407337
0.500	16	0.9957869	0.9303337	0.9413030
1.000	10	0.9953920	0.9258221	0.9367748
1.000	12	0.9954840	0.9279206	0.9389372
1.000	14	0.9955616	0.9271860	0.9382249
1.000	16	0.9956283	0.9271862	0.9380140

Les meilleurs paramètres sont donc 16 neurones et un coefficient de régularisation de 0.1.

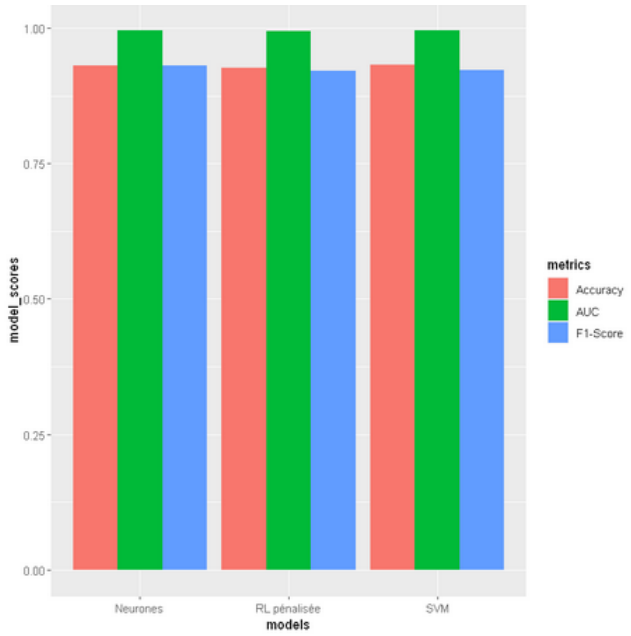
4.2 Comparaison des meilleurs modèles de chaque méthode

4.2.1 Comparaison des métriques globales

Afin de comparer les différents modèles précédemment construits, nous nous sommes appuyés sur trois critères que sont l'AUC (aire sous la courbe ROC), le F1 score et l'accuracy. L'AUC mesure la qualité des predicitions du modele, sans prendre en compte quel seuil de classification est choisi. Le F1-Score ou F-mesure combine la précision et le rappel. Vous trouverez ci-dessous un tableau récapitulatif des résultats :

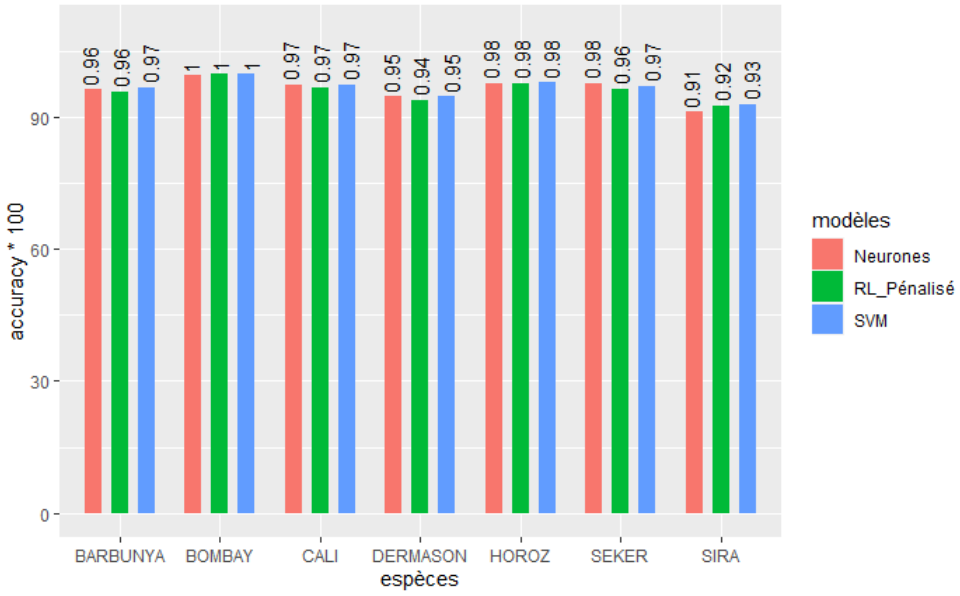
modèles	AUC	F1$_score$	Accuracy
RL pénalisée	0.9953801	0.9210867	0.9269608
SVM	0.9956398	0.9234694	0.9321078
Neurones	0.9964016	0.9309463	0.9316176

Le réseau de neurones a une AUC très légèrement meilleure que les autres ainsi que le meilleur F1 score (0.93). Les SVM ont le meilleur taux de reconnaissance (0.932).
Ces différences étant tellement minimes qu'elles se remarquent à peine graphiquement :



4.2.2 Comparaison des métriques par modalité

Nous avons utilisé une variante de l'accuracy décrite par cette formule : (TPR + TNR) /2 .
TPR : True positive rate TNR : True negative rate



	modèles	espèces	accuracy
1	RL_Pénalisé	BARBUNYA	0.9569226
2	RL_Pénalisé	BOMBAY	1.0000000
3	RL_Pénalisé	CALI	0.9659835
4	RL_Pénalisé	DERMASON	0.9387644
5	RL_Pénalisé	HOROZ	0.9755266
6	RL_Pénalisé	SEKER	0.9633193
7	RL_Pénalisé	SIRA	0.9248259
8	SVM	BARBUNYA	0.9658556
9	SVM	BOMBAY	1.0000000
10	SVM	CALI	0.9734194
11	SVM	DERMASON	0.9465144
12	SVM	HOROZ	0.9781133
13	SVM	SEKER	0.9696520
14	SVM	SIRA	0.9286003
15	Neurones	BARBUNYA	0.9620677
16	Neurones	BOMBAY	0.9967949
17	Neurones	CALI	0.9727232
18	Neurones	DERMASON	0.9464072
19	Neurones	HOROZ	0.9756862
20	Neurones	SEKER	0.9759429
21	Neurones	SIRA	0.9109557

La modalité "BOMBAY" est parfaitement reconnue par l'ensemble des modèles. Au contraire, la modalité "SIRA" est la moins reconnue par tous les modèles et connait des disparités entre ceux-ci. De plus, les modalités "CALI" et "HOROZ" sont reconnues de la même façon par tous les modèles.

5 Bilan du Projet

Pour finir, ce projet nous a permis d'approfondir nos compétences en apprentissage supervisé , ainsi que nos connaissances en matière de machine learning. Il aura été l'occasion de développer de réelles compétences en gestion de projet (répartition de tâches, gestion du temps) et donc d'appréhender la réalisation d'un projet au sein d'une équipe. Ces aspects seront réellement utiles dans le monde professionnel.