1 Objectif

Scilab et R – Performances comparées (Manipulation des données, Analyse discriminante).

Nous avons fait connaissance du logiciel Scilab dans un précédent tutoriel[1]. Nous étions arrivés à la conclusion qu'il se positionnait très bien comme une alternative à R dans le domaine du data mining même si, en matière de nombre de bibliothèques de méthodes statistiques et de data mining, il restait largement en retrait.

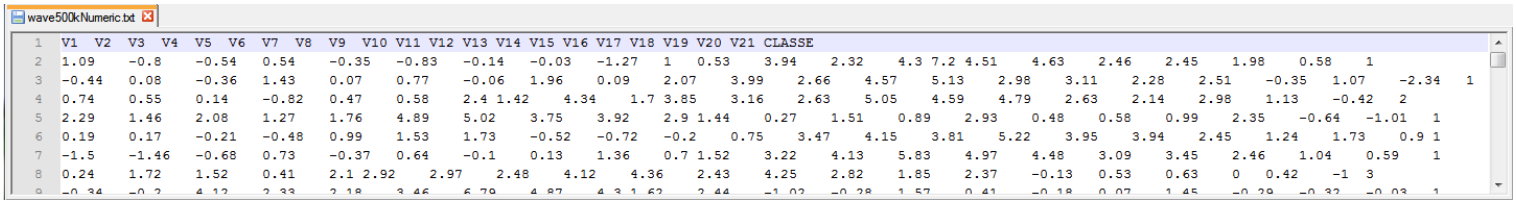
Dans ce second volet, nous nous intéressons aux performances de Scilab lors du traitement d'un grand fichier avec 500.000 observations et 22 variables. « Grand fichier » étant tout à fait relatif, nous le confrontons à la référence R pour mieux situer son comportement. Deux critères sont utilisés pour effectuer les comparaisons : l'occupation mémoire du logiciel mesurée dans le gestionnaire de tâches Windows, le temps de traitement à chaque étape du processus.

Il serait vain d'espérer obtenir une vision exhaustive. Pour délimiter notre champ d'étude, nous avons établi un scénario ultra-classique de data mining : charger un fichier de données, construire le modèle prédictif avec l'analyse discriminante linéaire, calculer la matrice de confusion et le taux d'erreur en resubstitution[2]. Bien évidemment, l'étude est forcément parcellaire. Il apparaît que Scilab semble moins à son avantage dans la gestion des données. Il est largement au niveau en revanche en ce qui concerne les traitements, cette dernière appréciation étant toutefois tributaire des packages ou toolbox utilisés.

2 Données

J'utilise souvent la base WAVEFORM dans mes comparatifs, en grande partie parce que nous disposons d'un générateur que nous pouvons configurer à souhait[3]. La variable cible CLASSE possède 3 modalités, nous disposons de 21 descripteurs numériques.

Dans ce tutoriel, nous avons généré une base de 500.000 observations. Le fichier « WAVE500KNumeric.txt » est au format texte (TSV, séparateur tabulation). Nous avons d'emblée codé la cible CLASSE en (1, 2, 3) dans le fichier. Voici un aperçu des premières observations.


wave500kNumeric.txt

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	CLASSE
2	1.09	-0.8	-0.54	0.54	-0.35	-0.83	-0.14	-0.03	-1.27	1	0.53	3.94	2.32	4.3	7.2	4.51	4.63	2.46	2.45	1.98	0.58	1
3	-0.44	0.08	-0.36	1.43	0.07	0.77	-0.06	1.96	0.09	2.07	3.99	2.66	4.57	5.13	2.98	3.11	2.28	2.51	-0.35	1.07	-2.34	1
4	0.74	0.55	0.14	-0.82	0.47	0.58	2.4	1.42	4.34	1.7	3.85	3.16	2.63	5.05	4.59	4.79	2.63	2.14	2.98	1.13	-0.42	2
5	2.29	1.46	2.08	1.27	1.76	4.89	5.02	3.75	3.92	2.9	1.44	0.27	1.51	0.89	2.93	0.48	0.58	0.99	2.35	-0.64	-1.01	1
6	0.19	0.17	-0.21	-0.48	0.99	1.53	1.73	-0.52	-0.72	-0.2	0.75	3.47	4.15	3.81	5.22	3.95	3.94	2.45	1.24	1.73	0.9	1
7	-1.5	-1.46	-0.68	0.73	-0.37	0.64	-0.1	0.13	1.36	0.7	1.52	3.22	4.13	5.83	4.97	4.48	3.09	3.45	2.46	1.04	0.59	1
8	0.24	1.72	1.52	0.41	2.1	2.92	2.97	2.48	4.12	4.36	2.43	4.25	2.82	1.85	2.37	-0.13	0.53	0.63	0	0.42	-1	3
9	-0.34	-0.2	4.12	2.33	2.18	3.46	6.79	4.87	4.3	1.62	2.44	-1.02	-0.28	1.57	0.41	-0.18	0.07	1.45	-0.29	-0.32	-0.03	1

3 Traitements avec Scilab

Nous avons programmé les traitements suivants sous Scilab.

[1] http://tutoriels-data-mining.blogspot.fr/2013/11/data-mining-avec-scilab.html

[2] On n'évalue pas vraiment les performances en prédiction ce faisant. Le taux d'erreur en resubstitution est souvent trop optimiste. Mais ça nous permet au moins de nous assurer que les systèmes produisent des modèles équivalents.

[3] Le générateur codé en R accompagne l'ouvrage de Hastie, Tibshirani et Friedman (2013, 10th printing) que nous pouvons consulter en ligne, http://www-stat.stanford.edu/~tibs/ElemStatLearn/

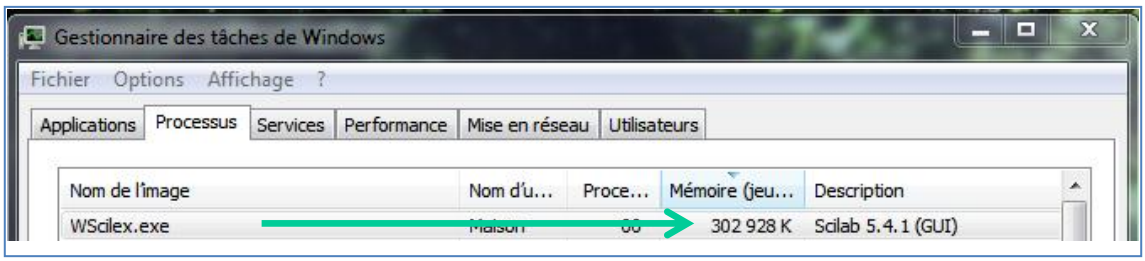
```
//augmenter la taille de la pile au maximum
stacksize("max")

//chargement des données
tic()
D=csvRead("wave500kNumeric.txt","\t",".","double",[],[],[2 1 500001 22])
disp(toc(),"duree chargement : ")

//variable cible y, descripteurs X
y=D(:,22:22)
X=D(:,1:21)

//distribution de la classe
disp(tabul(y),"distribution classe : ")

//construction du modèle
tic()
modele=nan_train_sc(X,y,'LD2')
disp(modele.weights,"Coefs. Analyse Discriminante")
disp(toc(),"duree apprentissage : ")

//prédiction
tic()
pred=nan_test_sc(modele,X)
disp(toc(),"duree prediction")

//matrice de confusion
mc=nan_confusionmat(y,pred.classlabel)
disp(mc,"matrice de confusion")

//erreur en resubstitution
disp(1.0-sum(diag(mc))/sum(mc),"taux erreur en resubstitution")
```

Quelques remarques importantes par rapport au code :

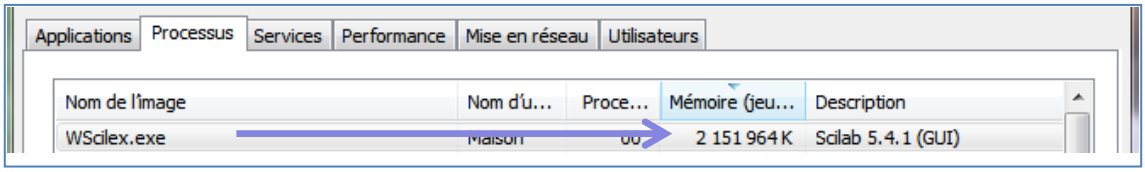
- La procédure tic() permet de lancer un chronomètre. toc() stoppe le chronomètre et renvoie l'écart de temps - en secondes - depuis l'appel du tic() précédent.
- Nous avons utilisé les procédures de la toolbox « nan » dans notre expérimentation. Nos résultats sont dépendants de ce choix.
- La matrice de confusion et le taux d'erreur en resubstitution servent avant tout à comparer notre modèle avec celui fourni par R dans la section suivante.

3.1 Chargement des données

Au démarrage de Scilab, son occupation mémoire est de 302.944 Ko, sachant que plusieurs librairies ont été automatiquement chargées.



La durée du chargement est de **25.887** secondes. L'occupation mémoire passe à 2.151.964 Ko (≈ 2,05 Go) [4]. Ce qui est considérable compte tenu de la taille pas si exceptionnelle de notre fichier. Cet aspect est quand même étonnant. D'autres tests ont été menés. Nous décrivons les résultats dans la conclusion.



Nous calculons la distribution des classes. Ce traitement est quasi-immédiat. Scilab affiche :

```
duree chargement :

  25.887

distribution classe :

   3.    166695.
   2.    166340.
   1.    166965.
```

Data loading time

Classes distribution

3.2 Construction du modèle

La phase d'apprentissage est extrêmement rapide (**2.497** sec.) avec la bibliothèque « NaN ».

```
Coefs. Analyse Discriminante

   0.5822831  - 0.5557168  - 0.5789095
 - 0.0005040    0.0002009  - 0.0000879
   0.0159417  - 0.0173581    0.0014031
   0.0311138  - 0.0332884    0.0015319
   0.0492916  - 0.0486050  - 0.0010144
   0.0659451  - 0.0667088    0.0000100
   0.0506132  - 0.0682628    0.0178066
   0.0326237  - 0.0678686    0.0351277
 - 0.0173101  - 0.0334772    0.0509946
 - 0.0666444  - 0.0005381    0.0672913
 - 0.0967968    0.0309974    0.0662301
 - 0.1299103    0.0650276    0.0655647
 - 0.0973207    0.0662260    0.0320581
 - 0.0653745    0.0666184  - 0.0010655
 - 0.0154794    0.0494523  - 0.0336744
   0.0315031    0.0338676  - 0.0658957
   0.0490389    0.0158864  - 0.0653032
   0.0664812    0.0012955  - 0.0676988
   0.0479959    0.0011973  - 0.0494664
   0.0343000  - 0.0007278  - 0.0333651
   0.0157122  - 0.0003111  - 0.0157114
 - 0.0020499    0.0011400    0.0007208

duree apprentissage :

  2.497
```

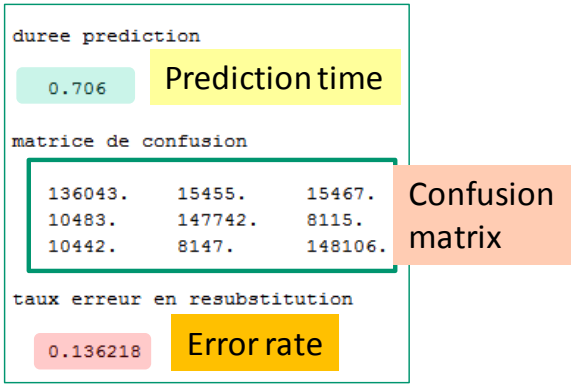
Processing time

[4] Les valeurs relevées (temps de traitement et occupation mémoire) peuvent légèrement varier d'une exécution à l'autre. Obtenir un ordre de grandeur est le plus important dans ce type de benchmark.

Il est difficile de se prononcer ou d'effectuer des comparaisons sans connaître avec exactitude la nature des calculs réalisés. L'occupation mémoire à l'issue de la construction du modèle est restée quasiment stable. C'est aussi une autre surprise.

3.3 Prédiction, matrice de confusion et taux d'erreur en resubstitution

Enfin, l'application du modèle sur les données pour produire la prédiction est également très rapide (**0.706** sec.). Nous obtenons un taux d'erreur en resubstitution de **13.62**%.



4 Traitements avec R

Nous avons effectué une séquence d'analyse équivalente sous R, en utilisant la procédure lda() du package MASS. Voici le code source de l'expérimentation.

```
#chargement des données
system.time(donnees <- read.table("wave500kNumeric.txt",sep="\t",dec=".",header=T))

#variable cible y, descripteurs X
y <- factor(donnees[,22])
X <- donnees[,1:21]

#distribution de la classe
print("distribution classe : ")
print(table(y))

#construction du modèle
library(MASS)
system.time(modele <- lda(X,y))
print("Modele")
print(modele)

#prédiction
system.time(pred <- predict(modele,newdata=X))

#matrice de confusion
mc <- table(y,pred$class)
print("matrice de confusion")
print(mc)

#erreur en resubstitution
```

```
print("taux erreur en resubstitution")
print(1.0-sum(diag(mc))/sum(mc))
```

Le schéma est le même. A la différence que nous devons transformer en type **factor** la variable cible avant de procéder à la construction du modèle.

Au démarrage de R, son occupation mémoire est de 31.188 Ko. Il est de 210.216 Ko après chargement, lequel a duré **12.07** sec.

```
> #chargement des données
> system.time(donnees <- read.table("D:/DataM
utilisateur     système      écoulé
      11.76        0.28       12.07
```

L'apprentissage prend **28.86** secondes. La situation est complètement inversée par rapport à Scilab.

```
> #construction du modèle
> library(MASS)
> system.time(modele <- lda(X,y))
utilisateur     système      écoulé
      27.42        1.19       28.86
> print("Modele")
[1] "Modele"
> print(modele)
Call:
lda(X, y)

Prior probabilities of groups:
      1       2       3
0.33393 0.33268 0.33339

Group means:
           V1            V2           V3           V4           V5        V6
1  0.001360046  0.4996380080  0.997329201  1.502305333  2.001211631 2.4988129
2 -0.002972466 -0.0006105567 -0.002567813 -0.001717025 -0.001783816 0.4974921
3 -0.002495096  0.5016525991  0.999614805  1.497237230  2.001382405 3.0009442
         V7       V8       V9       V10       V11       V12       V13       V14
1 2.998833 2.495864 1.995093 2.000862 2.003256 2.000217 2.000693 2.498886
2 1.000771 1.497511 1.999430 2.996692 4.002366 4.001771 3.999753 4.002890
3 4.002897 4.002709 4.002411 3.998685 3.997493 2.996838 1.999192 1.499308
        V15       V16          V17          V18          V19          V20
1 2.997297 2.5012555  2.004272093 1.499412991 1.0038480520 0.498647142
2 4.001393 2.9988358  2.002067332 1.503661356 0.9985835638 0.496575388
3 1.001745 0.5043985 -0.003656498 0.003106632 0.0006359519 0.002742674
           V21
1 -0.002415716
2  0.004578634
3  0.001181379

Coefficients of linear discriminants:
              LD1           LD2
V1  -0.0002713839 -0.0007058862
V2   0.0198299375  0.0287297577
V3   0.0356613735  0.0559545763
V4   0.0474855746  0.0895256623
V5   0.0675090141  0.1197865271
V6   0.0884903511  0.0917114011
V7   0.1057149539  0.0591139848
V8   0.0868160167 -0.0311007219
V9   0.0703531575 -0.1193812113
V10  0.0380758416 -0.1732499104
V11  0.0020755872 -0.2329852315
V12 -0.0340465489 -0.1749634170
V13 -0.0682462801 -0.1178689095
V14 -0.0851984066 -0.0285400934
V15 -0.1022878598  0.0558591284
V16 -0.0833633745  0.0877533002
V17 -0.0719837205  0.1192534404
V18 -0.0525720167  0.0853267875
V19 -0.0332335940  0.0620170944
V20 -0.0157362256  0.0280003751
V21 -0.0008562794 -0.0044970477

Proportion of trace:
   LD1    LD2
0.5404 0.4596
```

L'occupation mémoire est passée à 580.536 Ko. Gardons nous néanmoins de porter un jugement définitif sur cette étape. Il est très vraisemblable que les calculs soient menés différemment dans les deux logiciels. On le constate aisément en produisant la matrice de confusion et le taux d'erreur en resubstitution sous R (**13.59**%). Les modèles ne classent pas exactement de manière identique.

```
> #matrice de confusion
> mc <- table(y,pred$class)
> print("matrice de confusion")
[1] "matrice de confusion"
> print(mc)

y        1      2      3
  1 133836  16450  16679
  2   9340 148850   8150
  3   9214   8134 149347
>
> #erreur en resubstitution
> print("taux erreur en resubstitution")
[1] "taux erreur en resubstitution"
> print(1.0-sum(diag(mc))/sum(mc))
[1] 0.135934
```

La documentation de « NaN » étant très peu loquace au sujet de la méthode implémentée[5], il est dès lors difficile de porter un jugement arrêté sur les performances des systèmes respectifs.

5 Comparaisons

Pour compléter nos comparaisons, nous avons mené les mêmes expérimentations sous Tanagra et Sipina. Pour cette dernière, nous avons testé les versions monothread et multithread.

Logiciels	Durée chargement (sec.)	Durée modélisation (sec.)[6]	Occupation mémoire à l'issue de l'expérimentation (Mo)
Scilab ('LDA' de « NaN »)	25,89 sec.	2,49 sec.	2103,47 Mo
R (lda du package MASS)	12,07 sec.	28,86 sec.	566,93 Mo
Tanagra 1.4.49	3,96 sec.	5,29 sec.	56,06 Mo
Sipina Monothread	4,29 sec.	1,29 sec.	63,46 Mo
Sipina Multithread (4 threads)	4,29 sec.	0,38 sec.	64,07 Mo

Ce tableau nous inspire plusieurs commentaires :

- La version multithread de Sipina est réellement la plus rapide. Elle se positionne même avantageusement par rapport à la PROC DISCRIM de SAS sur plusieurs fichiers benchmark que nous avions testé dans un précédent tutoriel[7].

[5] Les calculs réalisés dans la procédure **lda**() de R est décrite dans l'ouvrage de Venables et Ripley, « Modern Applied Statistics with S », Springer, 2002 ; pages 331 à 338.

[6] Attention, les techniques sous-jacentes peuvent être différentes.

- Par rapport à Sipina, Tanagra introduit des calculs statistiques supplémentaires destinés à évaluer globalement le modèle et individuellement les variables prédictives. Elles grèvent tout naturellement le temps de traitement.
- Par rapport à R, Sipina et Tanagra mènent les calculs différemment. De fait, les temps de traitement ne seront pas non plus directement comparables. En revanche, les modèles classent de manière équivalente. Les matrices de confusion en resubstitution obtenues à l'issue des traitements sont strictement identiques.

Il semble que le problème de Scilab est avant tout l'occupation mémoire. La toolbox « NaN » est en revanche particulièrement efficace. On aurait aimé quand même obtenir plus de précisions quant à la nature des calculs réalisés pour l'estimation des coefficients des fonctions de classement de l'analyse discriminante.

6 Conclusion

Notre comparatif n'est pas exhaustif, loin de là. Je doute d'ailleurs que cela soit possible s'agissant de logiciels de data mining. Le plus simple est d'établir un scénario précis de traitements et de situer les différents outils dans ce canevas. Dans une démarche prédictive - construction d'un modèle et son application sur des observations - Scilab tient parfaitement sa place, avec peut être un petit bémol concernant l'occupation mémoire.

Pour préciser cette idée, nous avons évalué le comportement de Scilab avec d'autres bases de données. Nous avons repris les fichiers traités dans le tutoriel « Multithreading équilibré pour l'analyse discriminante » :

Fichier	Nombre d'observations [n]	Nombre de variables [p] (*incluant la cible*)	Nombre de valeurs (en millions) [n x p]	Taille du fichier (*.txt) sur le disque (Ko)	Occupation mémoire **après chargement** (Ko)
Wave500KLarge	500.000	122	61.00	361.577 Ko	2.885.760 Ko
Wave2M	2.000.000	22	44.00	184.864 Ko	2.587.212 Ko
Covtype	581.012	53	30.79	75.556 Ko	2.378.496 Ko
Mit_Face_Images	513.455	362	185.87	647.793 Ko	3.135.888 Ko

Notons qu'**un pic plus élevé est relevé durant les opérations de chargement**. C'est ce qui constitue la vraie limitation du logiciel pour la manipulation des données. Par exemple, **il est monté à 7 Go pour « mit_face_images »**. Ma machine équipée de 8 Go de RAM s'est mise à swapper intensément, gênant l'exécution des autres applications ouvertes sous Windows, et grevant très lourdement le

[7] http://tutoriels-data-mining.blogspot.fr/2013/06/multithreading-equilibre-pour-la.html (« Multithreading équilibré pour l'Analyse Discriminante »).

temps de traitement (près de 30 minutes pour cette base). Par la suite seulement, à la fin de la lecture du fichier, l'occupation mémoire se stabilise à la valeur relevée dans le tableau ci-dessus.

Curieusement, l'occupation mémoire des données n'augmente pas linéairement avec le nombre de valeurs dans le fichier. Cela semble indiquer que le mode de stockage va au-delà du simple tableau de réels à 2 dimensions en mémoire, ou encore que Scilab exploite de la mémoire virtuelle…

Je me suis arrêté à ce stade. A l'évidence, je ne maîtrise pas encore toutes les subtilités de Scilab pour pouvoir être réellement catégorique. Il reste que, pour Scilab comme pour les autres logiciels, l'occupation mémoire est un sujet d'importance dès lors que l'on souhaite manipuler des grands ensembles de données dans le data mining.