1 Topic

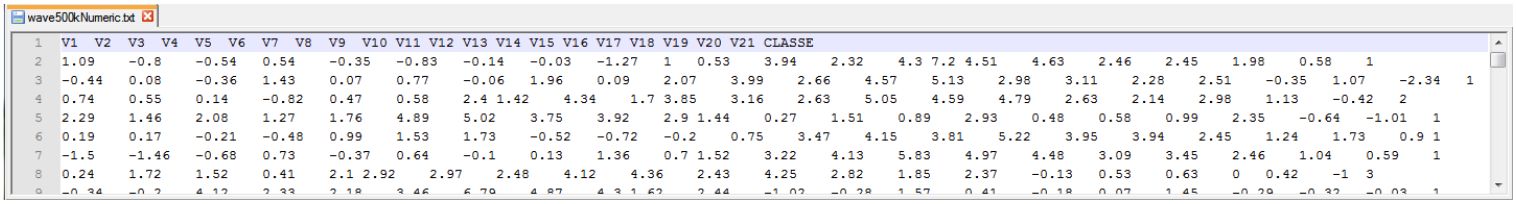
Scilab and R – Performance comparison in a supervised learning framework (memory occupation and processing time)

We have studied the Scilab tool in a data mining scheme in a previous tutorial[1]. We noted that Scilab is well adapted for data mining. It is a credible alternative to R. But, we observed also that the available toolboxes for statistical processing and data mining are not very numerous compared to those of R. In this second tutorial, we evaluate the behavior of Scilab when we deal with a dataset with 500,000 instances and 22 attributes. We compare its performances with those of R. Two criteria are used: the memory occupation measured in the Windows task manager; the execution time at each step of the process.

It is not possible to obtain an exhaustive point of view. To delimit the scope of our study, we have specified a standard supervised learning scenario: loading a data file, building the predictive model with linear discriminant analysis approach, calculating the confusion matrix and resubstitution error rate. Of course, this study is incomplete. But it seems that Scilab is less efficient in the data management step. It is however quite efficient in the modeling step. This last assessment depends on the toolbox used.

2 Dataset

I often use the WAVEFORM database, mainly because we have a generator that we can configure as we want[2]. There are 21 continuous predictors, the target variable is nominal (3 categories). In this tutorial, we generate 500,000 instances. We use the text file format (tab separated values). Here are the first rows of the data file “WAVE500KNumeric.txt “.


wave500kNumeric.txt

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	CLASSE
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
-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
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
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
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
-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
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
-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 Scilab

We programmed the following commands under Scilab.

```
//increasing the size of the stack
stacksize("max")

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

//target variable : y, predictors : X
```

[1] http://data-mining-tutorials.blogspot.fr/2014/01/data-mining-with-scilab.html

[2] http://www-stat.stanford.edu/~tibs/ElemStatLearn/

```
y=D(:,22:22)
X=D(:,1:21)

//frequency distribution of the target variable
disp(tabul(y),"distribution classe : ")

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

//prediction on the learning sample
tic()
pred=nan_test_sc(modele,X)
disp(toc(),"duree prediction")

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

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

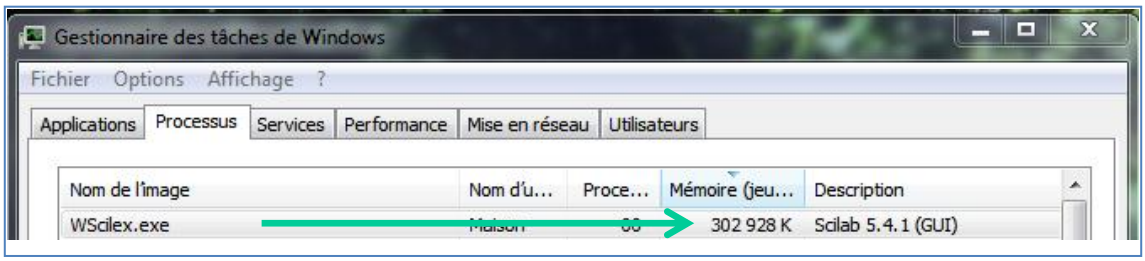
Some important comments about this program:

- The tic() command starts a stopwatch procedure; toc() stops the stopwatch and calculates the elapsed time since the previous tic() command.
- We use the "NaN" toolbox. Our results are dependents on this choice.
- The confusion matrix and the resubstitution error rate are mainly used to compare the results with those obtained with R.

We detail below the behavior of Scilab at every step of the processing.

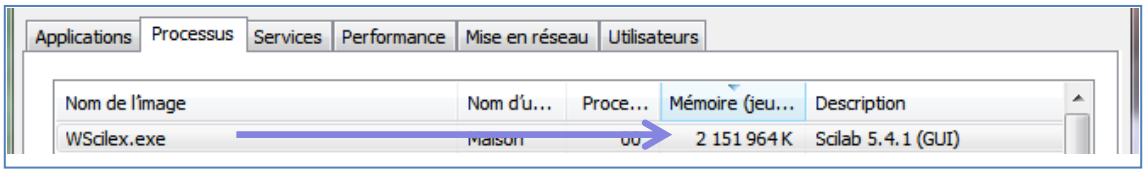
3.1 Data loading

When we start Scilab, the memory occupation is 302,928 KB, knowing that several toolboxes are loaded.



The loading time is **25.887** seconds. The memory occupation is 2,151,964 KB (≈ 2.05 GB)[3]. This value seems high if we consider the moderate size of the database. This is surprising. Other tests were led. We describe the results in the conclusion.

[3] The measured values (processing time and memory usage) may slightly vary from one execution to another.



We calculate the frequency distribution of the target variable. We obtain the following results.

```
duree chargement :

  25.887

distribution classe :

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

Data loading time

Classes distribution

3.2 Construction of the predictive model

The learning phase is extremely fast (**2.497** sec.) with the linear discriminant method of the "NaN" toolbox.

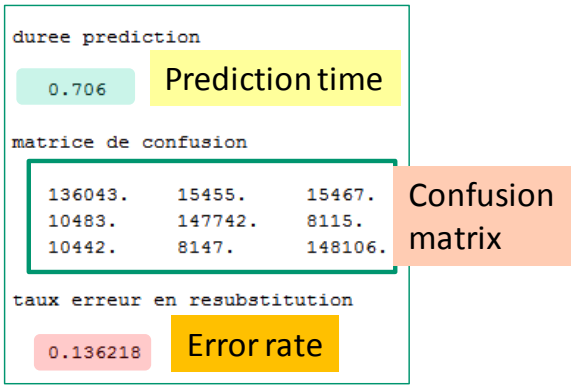
```
Coefs. Analyse Discriminante

   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

We cannot really make comparisons without knowing the exact nature of the calculations. The memory occupation after the construction of the model remained unchanged. This is also another surprise.

3.3 Prediction, confusion matrix and resubstitution error rate

We apply the model on the learning sample to obtain the prediction column (**0.706** sec.). The resubstitution error rate is **13.62**%.



4 R

R (http://www.r-project.org/) is a state-of-the art program in the data mining domain. It is always interesting to compare the behavior of a tool with the R software. Here is the source code for R.

```
#loading the dataset
system.time(donnees <- read.table("wave500kNumeric.txt",sep="\t",dec=".",header=T))

#target attribute y, predictors X
y <- factor(donnees[,22])
X <- donnees[,1:21]

#frequency distribution of the target variable
print("distribution classe : ")
print(table(y))

#learning phase
library(MASS)
system.time(modele <- lda(X,y))
print("Modele")
print(modele)

#prediction on the learning sample
system.time(pred <- predict(modele,newdata=X))

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

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

We have the same process. The difference is that we must transform the target column into the factor type under R. When we launch R, its memory occupation is 31,188 KB. It becomes 210,216 KB after the data loading. The loading time is **12.07** sec.

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

The learning process takes **28.86** seconds. The situation is inverted compared with Scilab for which the loading process is the most difficult.

```
> #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
```

The memory occupation becomes 580,536 KB. I think the two tools do not lead the calculations in the same way. Thus, the results (especially the computation time) are not really comparable. We note that the confusion matrix and the resubstitution error rate (**13.59%**) under R are different.

```
> #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
```

Because the documentation about the "NaN" toolbox does not describe the underlying approach, it is not really possible to understand the differences between the two tools[4].

5 Comparison with Tanagra and Sipina

To complete our comparison, we conducted the same experiments (linear discriminant analysis) with Tanagra and Sipina. For this latter, we tested the single-threaded and multithreaded versions.

Software	Data loading time (sec.)	Learning time (sec.)[5]	Memory occupation the experiment is achieved (MB)
Scilab ('LDA' from « NaN »)	25,89 sec.	2,49 sec.	2103.47 MB
R (lda from MASS package)	12,07 sec.	28,86 sec.	566.93 MB
Tanagra 1.4.49	3,96 sec.	5,29 sec.	56.06 MB
Sipina Monothread	4,29 sec.	1,29 sec.	63.46 MB
Sipina Multithread (4 threads)	4,29 sec.	0,38 sec.	64.07 MB

These results suggest some comments.

- The multithreaded version implemented into Sipina is really fast. It is favorably compared with the well known SAS program on several databases[6].
- Compared with Sipina, Tanagra implements a single-threaded approach. In addition, it computes other statistical indicators to evaluate the relevance of the whole model and each predictor. Thus, its execution time is less favorable.
- The underlying calculations are not the same between R and Sipina/Tanagra. The calculation time is not a good indicator here.

Ultimately, the main issue is memory usage for treatments that we want to achieve. It determines the ability of the software to perform the calculations, especially when the database size increases. It

[4] The underlying calculations of the lda() procedure for R are described in the Venables and Ripley's book, "Modern Applied Statistics with S", Springer, 2002 ; pp.331-338.

[5] The underlying calculations may be different.

[6] http://data-mining-tutorials.blogspot.fr/2013/09/load-balanced-multithreading-for-lda.html

seems that Scilab is not really efficient in this domain. The memory occupation can be large even if we deal with a moderate sized dataset. We explore this in detailed way below by processing various databases.

6 Conclusion

Scilab can perform a supervised learning process irrefutably. We had already observed this fact in a previous tutorial. But it seems here that the memory occupation becomes a critical issue when we handle a large database. To confirm this analysis, we observe the behavior of Scilab on various sized databases that we used in the tutorial about the multi-threaded discriminant analysis[7].

Dataset	Rows [n]	Variables [p] (including the target)	Number of values (million) [n x p]	Data file size (KB) [text file format]	Memory occupation (after data loading) [KB]
Wave500KLarge	500.000	122	61.00	361.577 KB	2.885.760 KB
Wave2M	2.000.000	22	44.00	184.864 KB	2.587.212 KB
Covtype	581.012	53	30.79	75.556 KB	2.378.496 KB
Mit_Face_Images	513.455	362	185.87	647.793 KB	3.135.888 KB

About the memory occupation, we note that a higher peak is observed during loading process. This is the real limitation of Scilab for the data manipulation. For instance, it increased up to 7 GB to "mit_face_images". My computer has 8 GB RAM had great difficulty, disturbing the execution of the other applications under Windows. The processing time was very long (almost 30 minutes for this dataset). Subsequently, at the end of the reading of the file, the memory occupation stabilizes at the value recorded in the above table.

Surprisingly, the memory occupation does not increase linearly with the database size. It seems that Scilab uses a more sophisticated strategy than simply copying the values in main memory. Clearly, more investigations are needed to better understand the behavior of Scilab in the data mining task.

[7] http://data-mining-tutorials.blogspot.fr/2013/09/load-balanced-multithreading-for-lda.html