1 Topic

Multithreaded implementation of linear discriminant analysis in SIPINA 3.10. Study of its impact on the execution time.

Most of the modern personal computers have multicore CPU. This increases considerably their processing capabilities. Unfortunately, the popular free data mining tools does not really incorporate the multithreaded processing in the data mining algorithms they provide, aside from particular case such as ensemble methods or cross-validation process. The main reason of this scarcity is that it is impossible to define a generic framework whatever the mining method. We must study carefully the sequential algorithm, detect the opportunity of multithreading, and reorganize the calculations. We deal with several constraints: we must not increase excessively the memory occupation, we must use all the available cores, and we must balance the loads on the threads. Of course, the solution must be simple and operational on the usual personal computers.

Previously, we implemented a solution for the decision tree induction in Sipina 3.5. We studied also the solutions incorporated in Knime and RapidMiner. We show that the multithreaded programs outperform the single-thread version. This is wholly natural. But we observed also that there is not a unique solution. The internal organization of the multithread calculations influences the behavior and the performance of the program[1]. In this tutorial, we present a multithreaded implementation for the linear discriminant analysis in **SIPINA 3.10**. The main property of the solution is that the calculation structure requires the same amount of memory compared with the sequential program. We note that in some situations, the execution time can be decreased significantly.

The linear discriminant analysis is interesting in our context. We obtain a linear classifier which has a similar classification performance to the other linear method on the most of the real databases, especially compared with the logistic regression which is really popular (Saporta, 2006 – page 480; Hastie et al., 2013 – page 128). But the computation of the discriminant analysis is comparably really faster[2]. We will see that this characteristic can be enhanced when we take advantage of the multicore architecture.

To better evaluate the improvements induced by our strategy, we compare our execution time with tools such as SAS 9.3 (proc discrim), R (lda of the MASS package) and Revolution R Community (an "optimized" version of R).

2 Linear discriminant analysis

There are many references which describe the linear discriminant analysis on the Net (e.g. https://onlinecourses.science.psu.edu/stat505/node/89). Below, we describe the main steps of the learning algorithm, those that may need a lot of resources.

[1] Tanagra, « Multithreading for decision tree induction » - http://data-mining-tutorials.blogspot.fr/2010/11/multithreading-for-decision-tree.html

[2] On the MIT FACE IMAGE dataset - see experiments - the SAS 9.3 logistic regression (proc logistic) does 7 min 08 sec while discriminant analysis (proc discrim) would not take more than 39.12 seconds!

The linear discriminant analysis deals with a classification problem. It aims to assign the instances described by a set of *p* quantitative measurements (X_1, X_2, ..., X_p) to a predefined group described by a categorical variable Y. There are K groups {1, 2,…, K}. We dispose of a learning sample of size *n* for the construction of the model. Let ω an instance, $y(\omega)$ correspond to the class value of this instance. The absolute frequency of the class *k* is n_k. Three steps may lead to an intense CPU usage:

1. Calculation of the K conditional covariance matrices of size (p x p).

$$S_k = \left(\frac{1}{n_k} \sum_{\omega : y(\omega)=k} \left[x_i(\omega) - \bar{x}_{i,k} \right] \times \left[x_j(\omega) - \bar{x}_{j,k} \right] \right)_{i,j=1,\cdots,p}$$

Where $\bar{x}_{i,k}$ is the conditional mean (for the k[th] group) of the variable X_i.

2. Calculation of the pooled covariance matrix $S = \frac{1}{n-K} \sum_{k=1}^{K} n_k \times S_k$
3. Inversion of the pooled covariance matrix (S^{-1}).

The step (2) does need to access the data. It does not raise major problem. Similarly, the inversion of S at the 3[rd] step is not really a bottleneck. Some linear algebra libraries can handle very efficiently this kind of calculation[3]. The 1[st] step is thus the main challenge for the acceleration of the calculations.

Sequential implementation. In a single-threaded programming, the simplest, and probably the fastest, is to perform only one passes on the data. This requires keeping in memory K matrices S_k of dimension (p x p). A rough estimation shows that the memory occupation remains contained on the most of databases[4]. It is even possible to further reduce their memory occupation since these matrices are symmetrical. In addition, this strategy presents the interest to be fully compatible with our optional storage of data on the disc when dealing with databases which cannot fit in main memory. The degradation of the processing time is almost imperceptible in this case.

Multithreaded implementation. In Sipina 3.10, we implement a simple solution for the linear discriminant analysis. In a first pass on the database, we create K indexes which enable to distinguish the group membership of the instances. Then we start a thread for each group (in the 2[nd] pass over the dataset). The main **advantage** of this approach is that the transformation of the sequential program in a multithreaded program is really easy. This solution does not require additional memory compared with the sequential implementation, except for the indexes. And the synchronization of the threads is also easy. We just wait that the last thread was completed before starting the calculation of the pooled covariance matrix S. There are two main **drawbacks** to this solution: (a) if the number of available cores is higher than K, the surplus is not used; (b) when we have imbalanced dataset, the execution time depends on the processing of the most frequent group (the group with the most important number of instances). The experiments will confirm these intuitions when we put in relation the execution time with the characteristics of the data.

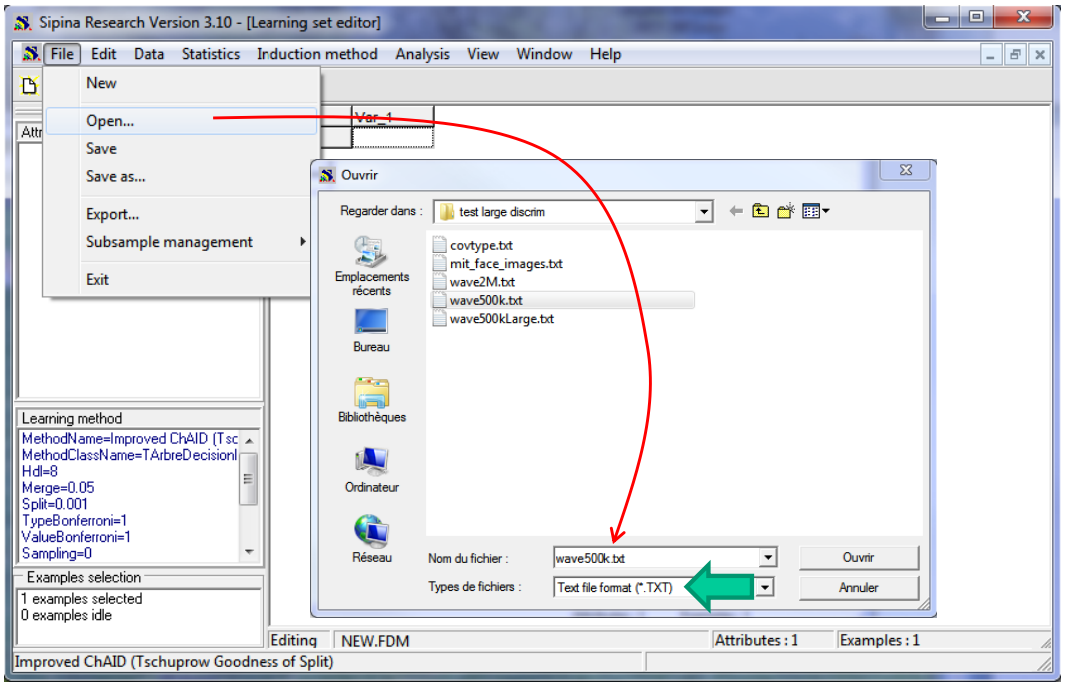
[3] Some libraries are really efficient e.g. http://www.netlib.org/lapack/

[4] For the MIT FACE IMAGE, K = 2 and p = 361. The memory occupation in double precision is [(361 x 361) x 8] x 2 ≈ 2 MB.

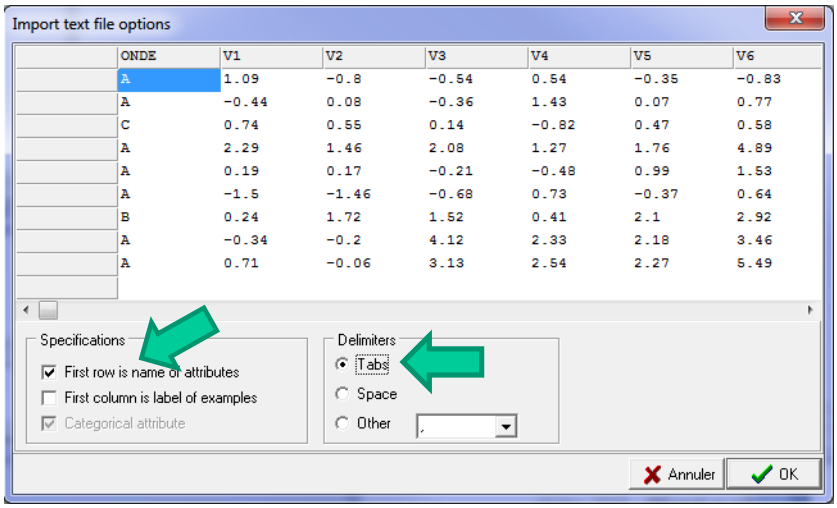
3 Linear discriminant analysis under SIPINA

3.1 Importing the dataset

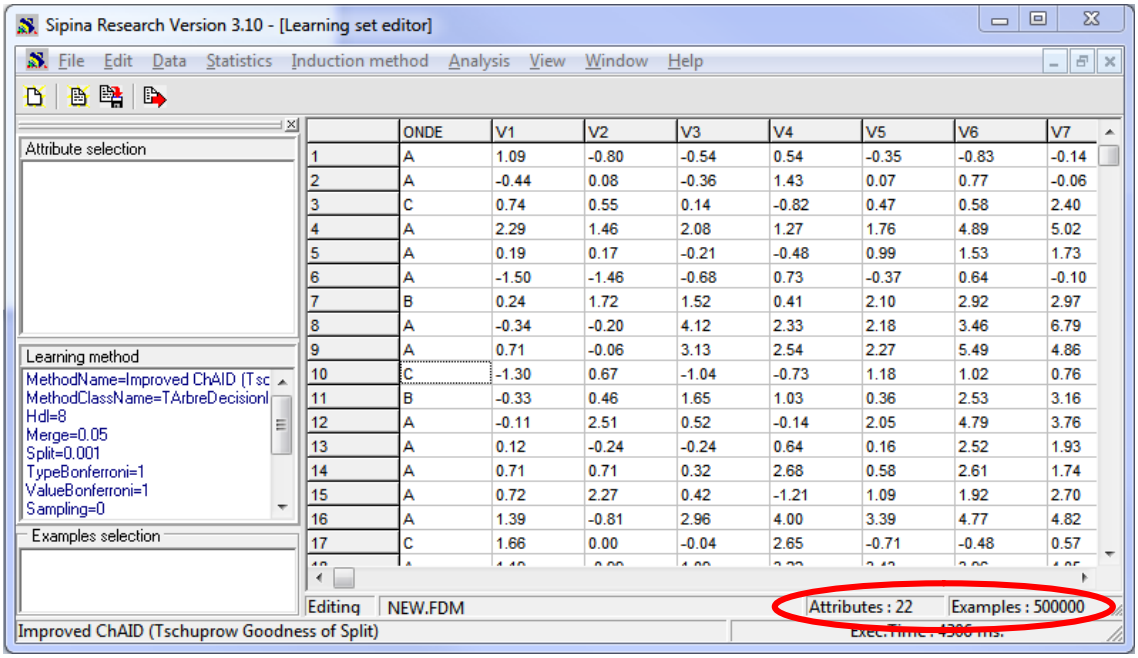
We use the Hastie and al. program (http://www-stat.stanford.edu/~tibs/ElemStatLearn/data.html ; **waveform.S**) to generate the WAVE500K database with n = 500,000 instances and p = 21 descriptors. There are K = 3 groups. After we launch SIPINA, we click on the FILE / OPEN menu, and we pick the WAVE500K.TXT data file (tab delimited text file format).



A wizard appears. It allows you to define the characteristics of the data file (type of delimiter, the first row corresponds to the names of the variables).

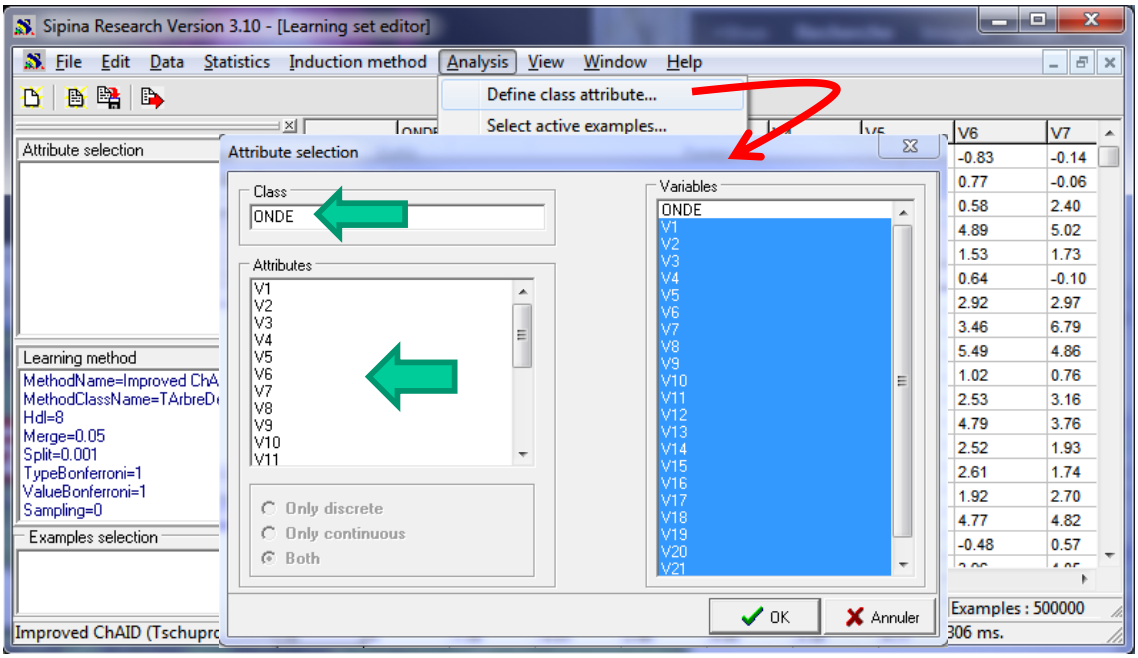


We confirm these specifications. The data file is imported and the values are visible into the Sipina data editor.



3.2 Defining the role of the variables

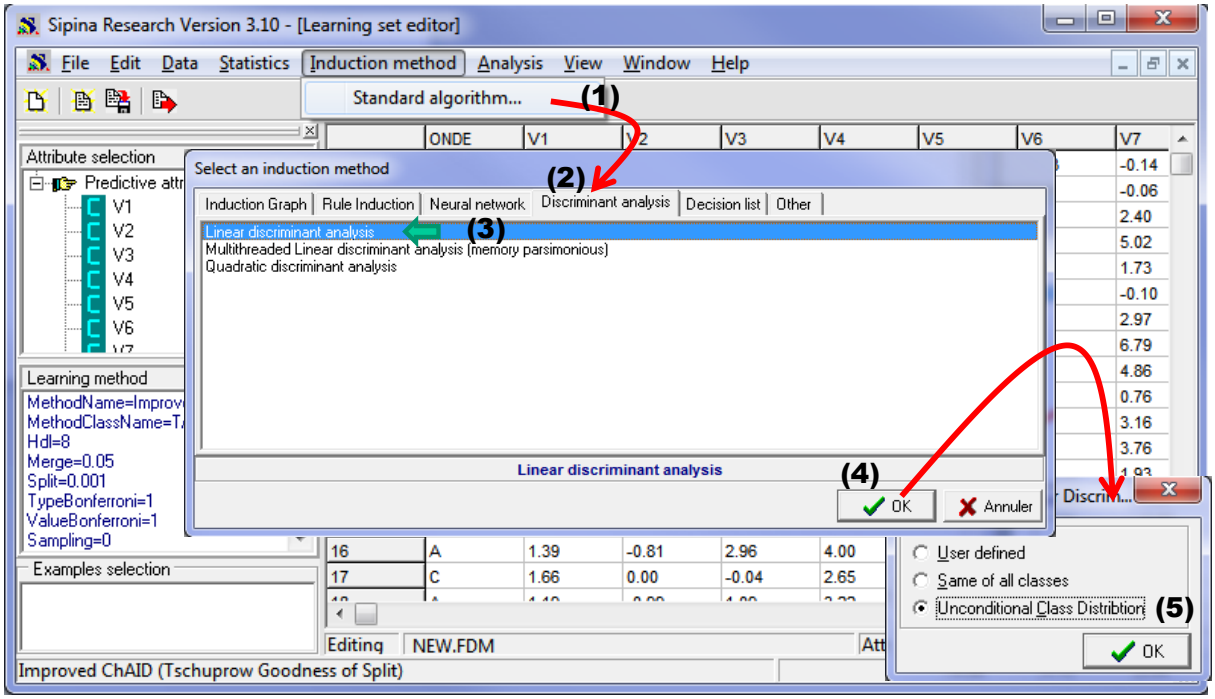
To specify the role of the variables, we click on the ANALYSIS / DEFINE CLASS ATTRIBUTE menu. By drag and drop, we set ONDE in CLASS, the other variables in ATTRIBUTES.



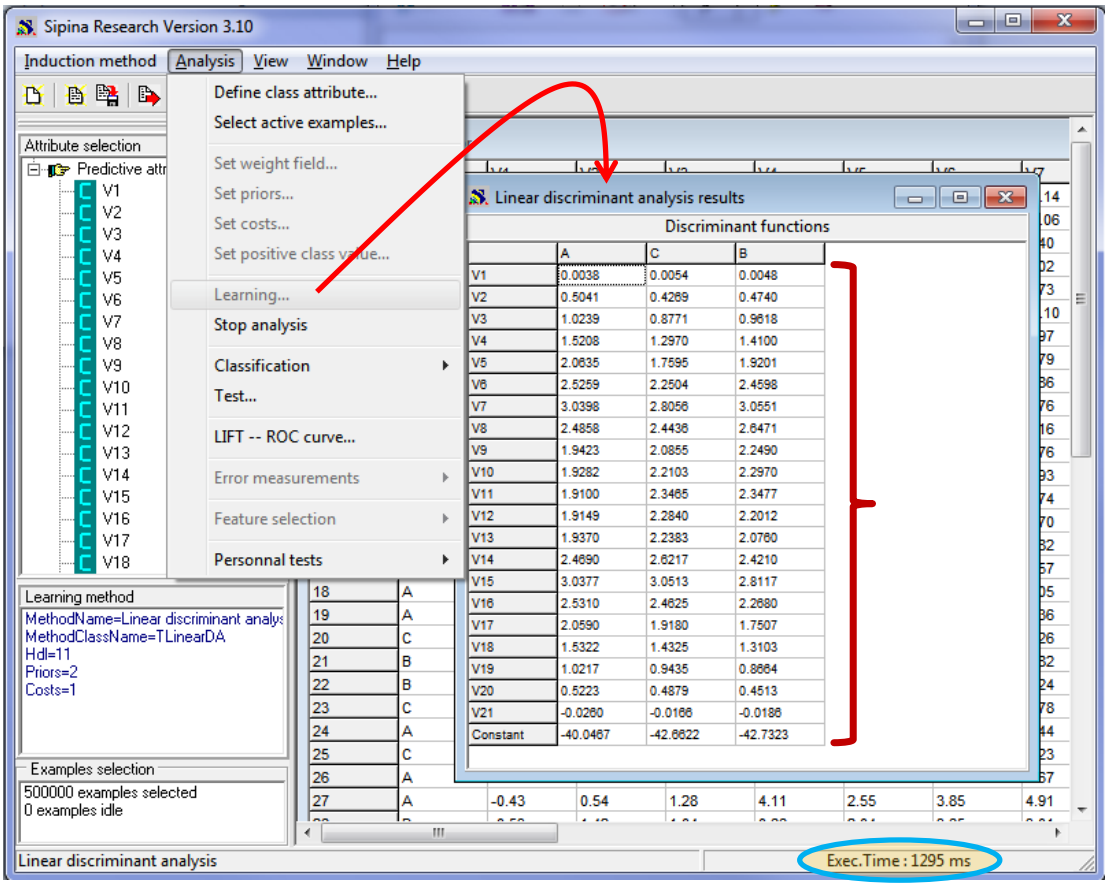
We confirm this choice. The selected variables appear in the left part of the main window.

3.3 Single-threaded implementation

We want to perform a linear discriminant analysis. To select the learning method, we click on the INDUCTION METHOD / STANDARD ALGORITHM menu.



Into the DISCRIMINANT ANALYSIS tab, we select the (single-thread) LINEAR DISCIMINANT ANALYSIS. We confirm the default settings (the priori class probabilities are estimated on the learning set). Then we click on the ANALYSIS / LEARNING menu to launch the learning process. The discriminant functions are displayed in a new visualization window after **1295** milliseconds.



By way of comparison, we show below the coefficients provided by SIPINA and SAS software. They are strictly identical.

Label	A	B	C
Constant	-40.04668	-42.73233	-42.66217
V1	0.00377	0.00476	0.00541
V2	0.50408	0.47400	0.42693
V3	1.02392	0.96181	0.87710
V4	1.52079	1.41004	1.29700
V5	2.06354	1.92008	1.75948
V6	2.52591	2.45984	2.25037
V7	3.03977	3.05506	2.80563
V8	2.48582	2.64713	2.44356
V9	1.94234	2.24896	2.08549
V10	1.92816	2.29702	2.21027
V11	1.91005	2.34766	2.34649
V12	1.91492	2.20122	2.28404
V13	1.93703	2.07603	2.23833
V14	2.46897	2.42098	2.62170
V15	3.03766	2.81172	3.05126
V16	2.53097	2.26796	2.46251
V17	2.05902	1.75070	1.91801
V18	1.53216	1.31029	1.43251
V19	1.02175	0.86640	0.94353
V20	0.52232	0.45131	0.48785
V21	-0.02602	-0.01864	-0.01655

SIPINA

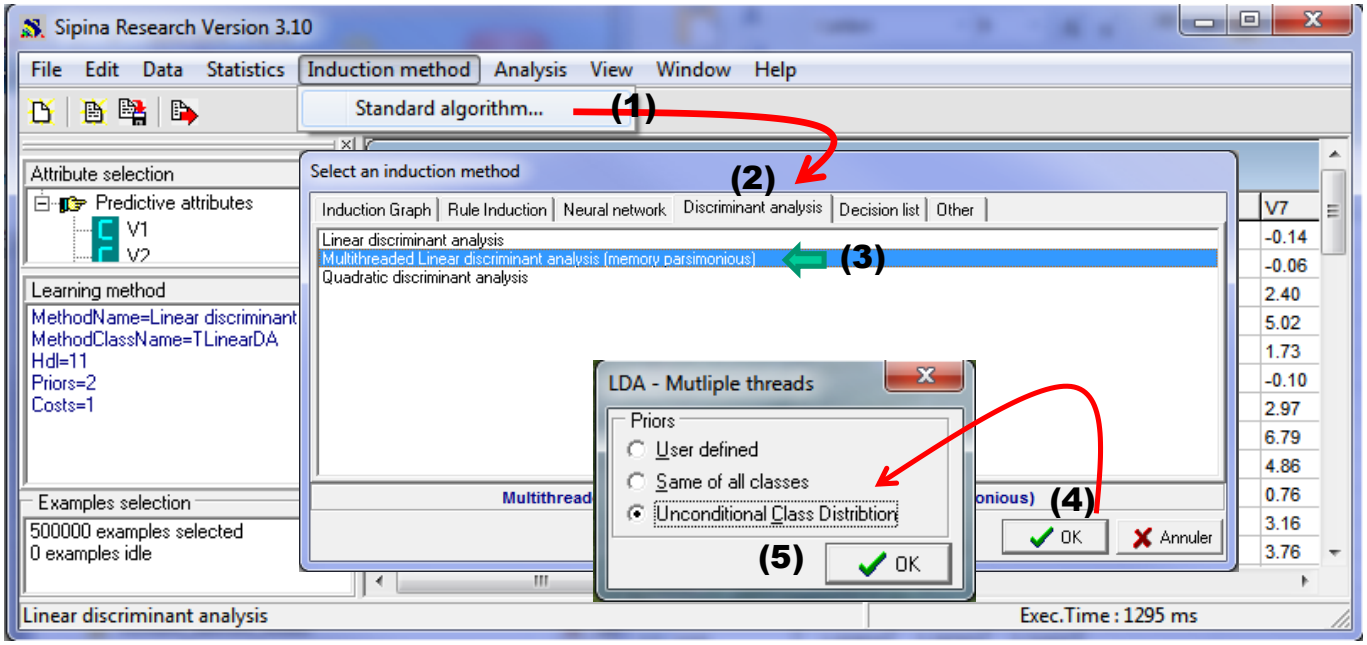
Linear Discriminant Function for ONDE

Variable	A	B	C
Constant	-40.04668	-42.73233	-42.66217
V1	0.00377	0.00476	0.00541
V2	0.50408	0.47400	0.42693
V3	1.02392	0.96181	0.87710
V4	1.52079	1.41004	1.29700
V5	2.06354	1.92008	1.75948
V6	2.52591	2.45984	2.25037
V7	3.03977	3.05506	2.80563
V8	2.48582	2.64713	2.44356
V9	1.94234	2.24896	2.08549
V10	1.92816	2.29702	2.21027
V11	1.91005	2.34766	2.34649
V12	1.91492	2.20122	2.28404
V13	1.93703	2.07603	2.23833
V14	2.46897	2.42098	2.62170
V15	3.03766	2.81172	3.05126
V16	2.53097	2.26796	2.46251
V17	2.05902	1.75070	1.91801
V18	1.53216	1.31029	1.43251
V19	1.02175	0.86640	0.94353
V20	0.52232	0.45131	0.48785
V21	-0.02602	-0.01864	-0.01655

SAS

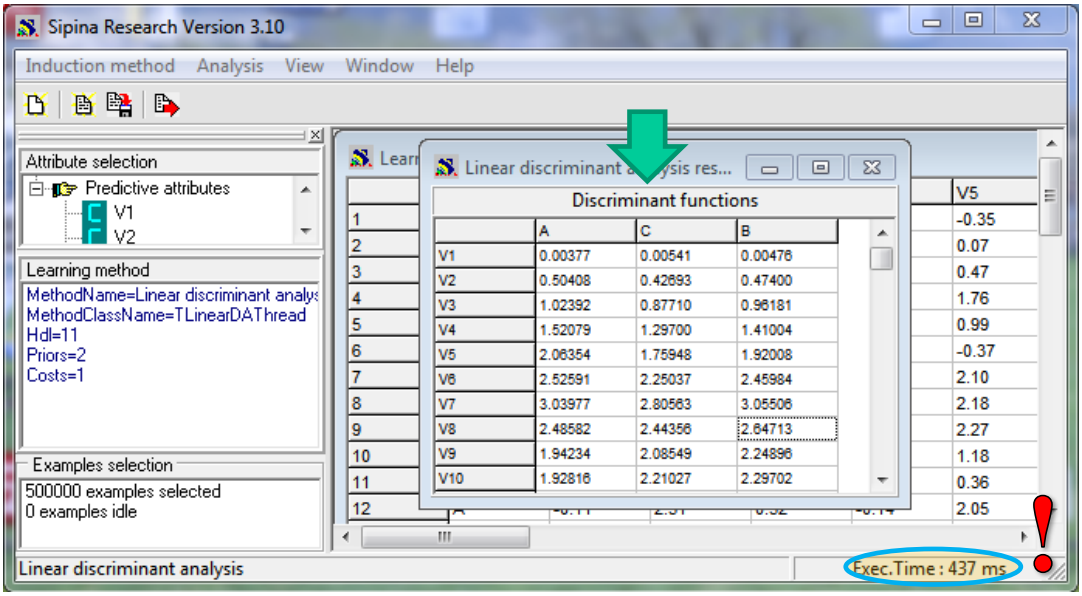
3.4 Multithreaded implementation

We must stop the current analysis. We do this by clicking on the ANALYSIS / STOP ANALYSIS menu. Then, we select the new learning algorithm (INDUCTION METHOD / STANDARD ALGORITHM…).

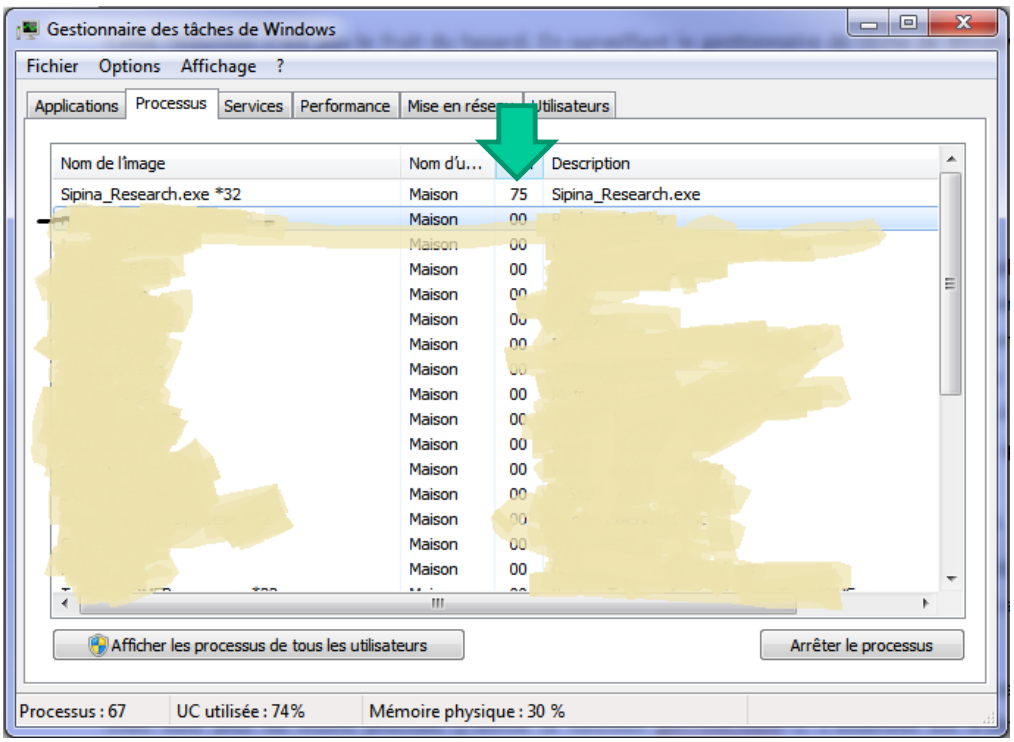


We select MULTITHREADED LINEAR DISCRIMINANT ANALYSIS (MEMORY PARSIMONIOUS) in the DISCRIMINANT ANALYSIS tab. We have the same settings as previously.

We launch again the calculations by clicking on the ANALYSIS / LEARNING menu. We obtain the same coefficients as the single-thread version. But the execution time is now **437** ms.



This reduction is not the result of chance. My machine has a quad-core processor (Q9400, 4 cores). By monitoring the Windows Task Manager, we note that K = 3 hearts among the 4 (75%) are used during the calculations.



3.5 Experiments on various databases

To evaluate the behavior of our multithreaded solution in comparison with the sequential implementation, we measured the execution time on various databases. For the WAVE datasets, we have always K = 3 balanced classes: WAVE500K that we described previously; WAVE500KLARGE with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n =

2,000,000 instances and p = 21 descriptors. These are all artificial databases that we can change at our discretion. The idea is to study the impact of the evolution of the number of instances and the number of descriptors compared with the WAVE500K.

We have also process the COVTYPE[5] dataset with K = 7 classes, but rather unbalanced (2 classes of the target variable concentrated a large part of the observations); and MIT FACE IMAGE[6] dataset, with K = 2, but with very unbalanced classes.

In the table below, we show the execution time in seconds[7]. "Ratio" indicates the reduction of execution time when we use the multithreaded version (e.g. 3.0 = 1.30 / 0.44). Because we use a quad-core processor, the best possible value for ratio is 4. When the number of groups (K) is lower than 4, the best value should be K.

Dataset	K	n	p	SIPINA (std)	SIPINA (threads)	Ratio
Wave 500k	3	500000	21	1.30	0.44	**3.0**
Wave 500k Large	3	500000	121	19.19	6.13	**3.1**
Wave 2M	3	2000000	21	5.16	1.83	**2.8**
Covtype	7	581012	52	5.30	2.67	**2.0**
Face Images	2	513455	361	142.73	135.46	**1.1**

The results suggest:

- The multithreaded approach always improves the execution time. The construction of the additional indexes for recognizing the class membership of the instances does not put a disadvantage.

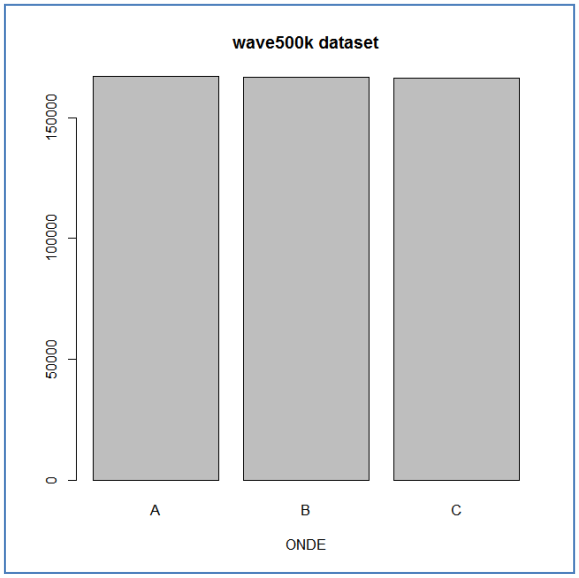


Figure 1 – Class values distribution for "wave500K"

[5] http://archive.ics.uci.edu/ml/datasets/Covertype

[6] http://c2inet.sce.ntu.edu.sg/ivor/cvm.html (Extended MIT face + non-face images data set).

[7] Strictly, it would be necessary to repeat the experiments and compute average of execution times.

- Compared with WAVE500K, the solution is relevant when we increase the number of variables (WAVE500KLarge) or the number of observations (WAVE2M). But we are in a particularly favorable configuration here. Firstly, the number of classes K = 3 is close to the number of cores in my machine (quad-core). On the other hand, we have a balanced dataset (Figure 1). The loads on the cores are well balanced until the end of the calculation of the conditional covariance matrices. So, we divide by about 3 the processing time since K = 3 cores are used.
- The situation is less favorable for COVTYPE. Yet, with K = 7 classes, we should fully use the potential of the machine. But the reduction ratio is 2. We better understand this result when we compute the class-attribute distribution (Figure 2). The execution time relies mainly of the processing of the groups "spruce" and "lodgepole". The calculation of the pooled covariance matrix is only possible when we have terminated the processing of the "lodgepole" group.

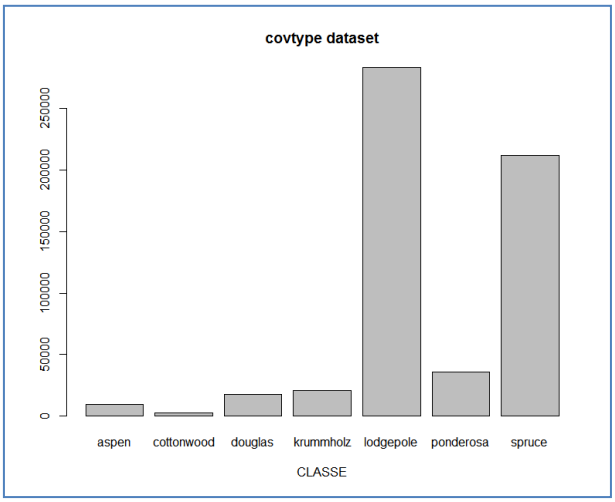


Figure 2 – Class values distribution for "covtype"

- The result is definitely disappointing for the MIT FACE IMAGE database. We understand why when we study the class-attribute distribution. In fact, only one core is used. The processing of the "pos" group is very quickly completed (Figure 3).

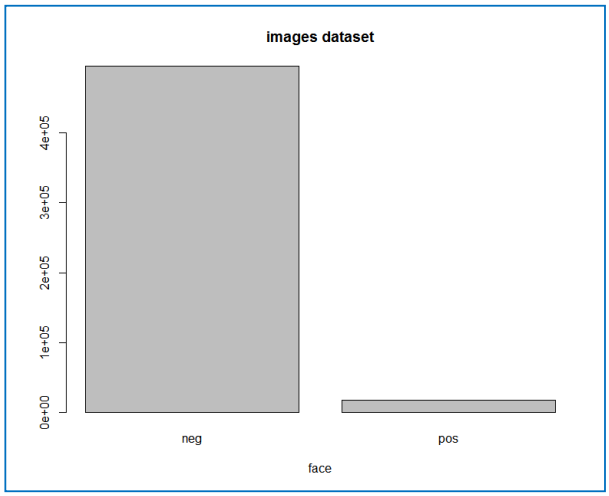


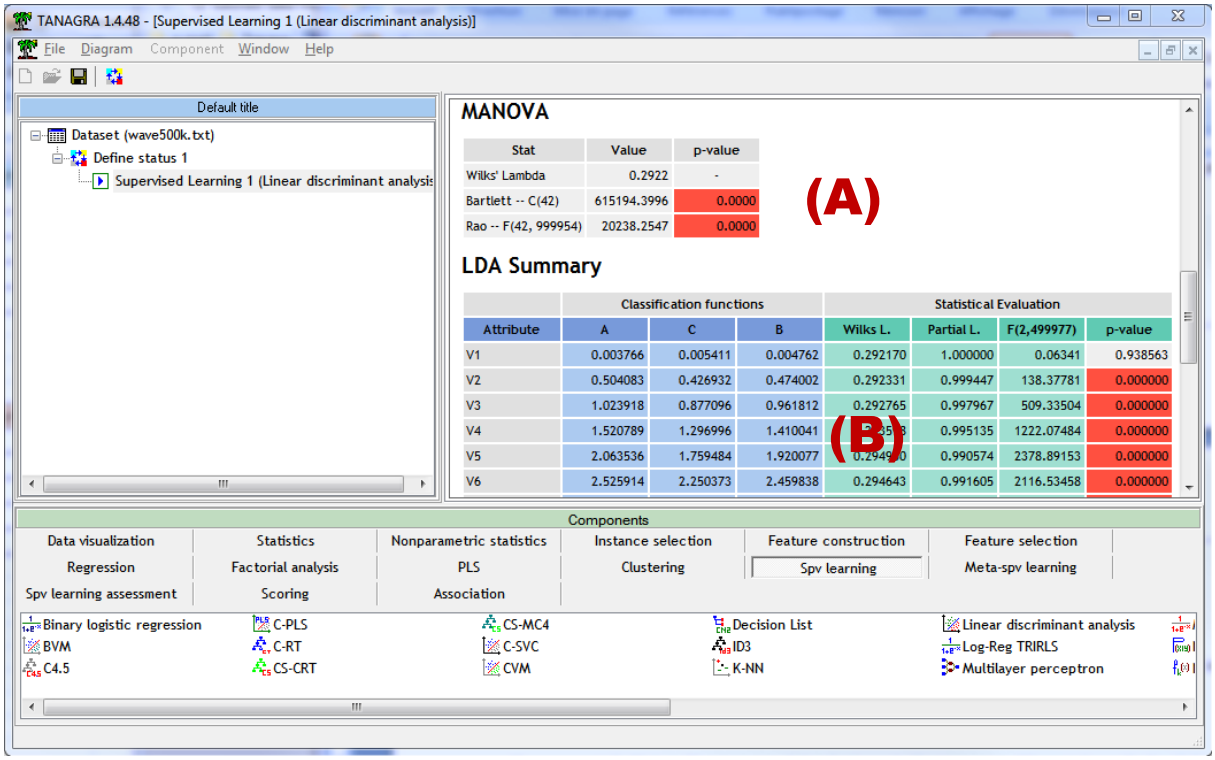
Figure 3 – Class value distribution for "mit face images"

Conclusion. Better use the capabilities of the machine, here by loading the available cores, is beneficial in terms of performance. In all cases, we have improved the execution time, including in the most unfavorable configuration (MIT_FACE_IMAGE). But it is also clear that we can do better. We will talk about it in the general conclusion.

4 Comparison with the other tools

4.1 Comparison with Tanagra

The linear discriminant approach is available in Tanagra. We show below the screenshot for the processing of the WAVE500K dataset[8].



Compared with Sipina, Tanagra produces additional indicators which require additional calculations: **(A)** the Wilks' lambda enables to evaluate the relevance of the whole model, we need to calculate the global covariance matrix and the determinant of both the global and the pooled covariance matrices; **(B)** the partial lambda enables to evaluate the relevance of the variables, here also we need to calculate various version of the covariance matrices and their determinant. So, the processing time for the WAVE500K is 5.25 sec. under Tanagra.

4.2 Comparison with SAS, R and Revolution R

We have measured the execution time of SAS 9.3 (proc discrim), R 3.0.0 (64 bits) and Revolution R Community 6.2.0 (64 bits) on the same databases.

For SAS, we use the following program for the WAVE500K dataset. No optional calculation is asked. I do not know if SAS produces internal objects which need additional computation.

[8] See http://data-mining-tutorials.blogspot.fr/2012/11/linear-discriminant-analysis-tools.html for the use of this approach in Tanagra and other tools.

```
proc discrim data = mesdata.wave500k;
   class onde;
   priors proportional;
run;
```

For R and Revolution R, we use the **lda()** procedure from the MASS package:

```
wave500k <- read.table(file="wave500k.txt",header=T,dec=".",sep="\t")
system.time(model.1 <- lda(ONDE ~ ., data = wave500k))
print(model.1)
```

We collect execution times in the following table. We placed the tools from left to right depending on their performance:

Dataset	K	n	p	SIPINA (threads)	SAS	SIPINA (std)	Revol. R 6.2.0	R 3.0.0
Wave 500k	3	500000	21	**0.44**	1.65	1.30	33.89	37.68
Wave 500k Large	3	500000	121	**6.13**	9.09	19.19	187.29	264.89
Wave 2M	3	2000000	21	**1.83**	6.19	5.16	137.05	148.42
Covtype	7	581012	52	**2.67**	4.29	5.30	67.29	84.69
Face Images	2	513455	361	135.46	**39.12**	142.73	ERR	ERR

We note that:

- Obviously, the multithreaded approach is efficient. The multithreaded of Sipina is better than SAS in 4 datasets on 5.
- The MIT FACE IMAGES database is particular. Firstly, we know that we have only two groups and they are very unbalanced. Secondly, some previous experiments on various learning methods show that the internal structure of Sipina (the data are stored in columns) are better adapted to some algorithms (e.g. the decision tree induction[9]) than the others (e.g. support vector machine[10], all the methods where we need to calculate scalar products). This dataset is the one which has the higher number of descriptors (p = 361) in our experiments, the performance is less good than SAS because of this structure. We note however that SIPINA can achieved the computation unlike R and Revolution R Community.
- R and Revolution R seem definitely less efficient. This is mainly because they use another calculation framework. It seems that the discriminant functions are deducted from a singular value decomposition process (Venables & Ripley, 2002; page 334). In fact, the processing times are not comparable to SIPINA or SAS.
- Revolution R Community is always faster than R. But the gaps are not as spectacular as those shown on the editor's website. I have already noticed this kind of results in a previous paper[11].
- The calculations are not achieved successfully under R and Revolution R for the MIT FACE IMAGES database. They both send the following error message: "Reached total allocation of 8127Mb".

[9] http://data-mining-tutorials.blogspot.fr/2011/10/decision-tree-and-large-dataset-follow.html

[10] http://data-mining-tutorials.blogspot.fr/2009/07/implementing-svm-on-large-dataset.html, the internal structures of SIPINA and TANAGRA are similar.

[11] http://data-mining-tutorials.blogspot.fr/2012/07/revolution-r-community-50.html

Because of its internal structure, SIPINA is less efficient than SAS when the number of descriptors increases (e.g. “Mit Face Images”, “Wave500KLarge” in the single-threaded process). In contrast, even more so with the multithreaded strategy, Sipina is clearly better when n >> p (e.g. Wave2M).

5 Conclusion

The parallelization of the data mining algorithms is not a new field. Many works exist. But they often correspond to specific solutions for some architecture. They rarely come out of the laboratories and do not reach the generalist tools. We note that many efforts are lead under R for the "high performance computing"[12]. But, they often correspond to the set up of environments allowing programming parallel algorithms rather than modifications of existing methods.

In this tutorial, we present a simple solution for discriminant analysis. It takes advantage of additional performance that offers multicore processors computers. Compared to the sequential program, the memory occupation of the calculation structure is unmodified. This is a great advantage for the processing of very large databases. We have implemented the solution in **SIPINA 3.10**. The experiments show that it allows reducing in significant proportions the execution time, especially for some datasets.

But the experiments show also that we can do better. To obtain an efficient behavior whatever the dataset that we deal, we must use all the available cores on the machine i.e. the number of used cores is a setting of the algorithm. In addition, we must also achieve a better load balancing among the cores.

6 References

Hastie T., Tibshirani R., Friedman J., « The Elements of Statistical Learning », Springer, 10th printing, January 2013.

Saporta G., « Probabilités, Analyse de Données et Statistique », Technip, 2006.

Venables W.N., Ripley B.D., « Modern Applied Statistics with S », Springer, 2002.

[12] http://cran.r-project.org/web/views/HighPerformanceComputing.html