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 program¹. 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; <u>Hastie et al.</u>, 2013 – page 128). But the computation of the discriminant analysis is comparably really faster². 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), <u>R</u> (Ida of the MASS package) and <u>Revolution R Community</u> (an "optimized" version of R).

2 Linear discriminant analysis

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

R.R.

¹ Tanagra, « Multithreading for decision tree induction » - <u>http://data-mining-tutorials.blogspot.fr/2010/11/multithreading-</u> 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, $\gamma(\omega)$ 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 kth 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 3rd step is not really a bottleneck. Some linear algebra libraries can handle very efficiently this kind of calculation³. The 1st 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 databases⁴. 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 2nd 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.

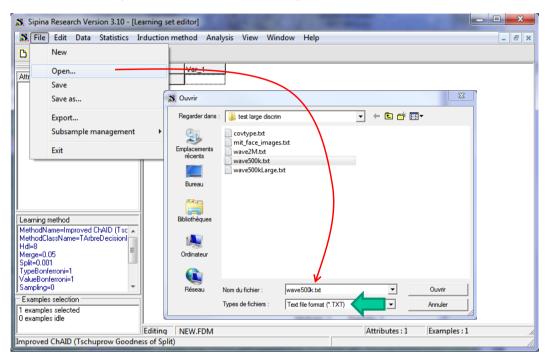
³ Some libraries are really efficient e.g. <u>http://www.netlib.org/lapack/</u>

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 \approx 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).

	ONDE	V1	V2	V3	V4	V5	V6
	A	1.09	-0.8	-0.54	0.54	-0.35	-0.83
	A	-0.44	0.08	-0.36	1.43	0.07	0.77
	с	0.74	0.55	0.14	-0.82	0.47	0.58
	A	2.29	1.46	2.08	1.27	1.76	4.89
	A	0.19	0.17	-0.21	-0.48	0.99	1.53
	A	-1.5	-1.46	-0.68	0.73	-0.37	0.64
	в	0.24	1.72	1.52	0.41	2.1	2.92
	A	-0.34	-0.2	4.12	2.33	2.18	3.46
	A	0.71	-0.06	3.13	2.54	2.27	5.49
Specifica	ations		Delimiters				
🗸 First i	row is name of a	attributes	Tabs				
First	column is label (of examples	C Space				
🗸 Cate	gorical attribute		C Other		•		

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

Eile Edit Data Statistics	Induction n	nethod <u>A</u> n	ialysis <u>V</u> iev	v <u>W</u> indow	<u>H</u> elp				- 5
🖸 🖹 📑 🕞									
12	4	ONDE	V1	V2	V3	V4	V5	V6	V7
.ttribute selection	. 1	A	1.09	-0.80	-0.54	0.54	-0.35	-0.83	-0.14
	2	A	-0.44	0.08	-0.36	1.43	0.07	0.77	-0.06
	3	С	0.74	0.55	0.14	-0.82	0.47	0.58	2.40
	4	A	2.29	1.46	2.08	1.27	1.76	4.89	5.02
	5	A	0.19	0.17	-0.21	-0.48	0.99	1.53	1.73
	6	A	-1.50	-1.46	-0.68	0.73	-0.37	0.64	-0.10
Learning method MethodName=Improved ChAID (Tsc A MethodClassName=TArbreDecisionI	7	В	0.24	1.72	1.52	0.41	2.10	2.92	2.97
	8	A	-0.34	-0.20	4.12	2.33	2.18	3.46	6.79
	9	A	0.71	-0.06	3.13	2.54	2.27	5.49	4.86
	10	С	-1.30	0.67	-1.04	-0.73	1.18	1.02	0.76
	11	В	-0.33	0.46	1.65	1.03	0.36	2.53	3.16
IdI=8 1erae=0.05	12	A	-0.11	2.51	0.52	-0.14	2.05	4.79	3.76
plit=0.001	13	A	0.12	-0.24	-0.24	0.64	0.16	2.52	1.93
Spill=0.001 TypeBonferroni=1 ValueBonferroni=1 Sampling=0	14	A	0.71	0.71	0.32	2.68	0.58	2.61	1.74
	15	A	0.72	2.27	0.42	-1.21	1.09	1.92	2.70
	16	A	1.39	-0.81	2.96	4.00	3.39	4.77	4.82
	17	С	1.66	0.00	-0.04	2.65	-0.71	-0.48	0.57
	40	1.	4.40	0.00	4.00	2.00	0.40	2.00	4.05
						A	ibutes : 22	F 1	500000
nproved ChAID (Tschuprow Goodn	Editing	NEW.FDM				Att		Examples	: 500000

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.

Sipina Research Version 3.10 - [Learning set editor]			x
Statistics Induction method Analysis View Window Help			- 8 ×
🖸 📓 🖳 Define class attribute			
	1/5	V6	V7 🔺
Attribute selection	23	-0.83	-0.14
Variables		0.77	-0.06
		0.58	2.40
		4.89	5.02
		1.53	1.73
Attributes V3 V1 V4 V2 V6 V3 V7		0.64	-0.10
		2.92	2.97
		3.46	6.79
Learning method		5.49	4.86
MethodName-Improved ChA	=	1.02	0.76
Method/lassName=TArbreD/		2.53	3.16
		4.79	3.76
		2.52	1.93
TypeBonferroni=1		2.61	1.74
ValueBonremoni=1		1.92	2.70
V18		4.77	4.82
Examples selection C Only continuous V19 V20		-0.48	0.57
Both V20 V21	-	2.00	4.00
			P
🗸 ОК 🗙	Annuler II	Examples : 5	00000 //
Improved ChAID (Tschuprc		306 ms.	

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.

Sipina Research Ver								
<u> F</u> ile <u>E</u> dit <u>D</u> ata	Statistics	Induction method	<u>A</u> nalysis <u>V</u> iew	<u>W</u> indow	<u>H</u> elp			- 8 :
🔁 🛛 🏝 🗳		Standard algo	rithm	D				
	X		E V1	V2	V3	V4	V5 V6	6 V7 -
Attribute selection	elect an inductio	on method	(2)					-0.14
Predictive attr	Industion Graph	Rule Induction Neur	al naturatik Discrimin	ant analysis De	oision list Otho	•]		-0.06
	Linear discriminar	' '	(3)		cision list Othe	:		2.40
		nt analysis lear discriminant analysis	(memory parsimoniou:	s]				5.02
	Quadratic discrimi	iinant analysis						1.73
- C V5								-0.10
V6								2.97
1/7								6.79
Learning method								4.86
MethodName=Improv								0.76
MethodClassName=T/ Hdl=8								3.16
Merge=0.05			Linear disc	riminant analys	zie	(4)	/	3.76
Split=0.001			Enfour disc	initiant analys		(4)		Discrit
TypeBonferroni=1 ValueBonferroni=1						🗸 🗸 oʻ	K Annuler	
Sampling=0	T	16 A	1.39	-0.81	2.96	4.00	O User defined	-
Examples selection		17 C	1.66	0.00	-0.04	2.65	O Same of all class	
		40	4.40	0.00	4.00	2.00		Class Distribtion (5
							··· onconuitional g	
		Editing NEW.	DM			Att		🗸 ок
mproved ChAID (Tschi	uprow Goodne	ess of Split)				Ĺ		

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.

Define clas									
	s attribute			\frown					
Select activ	e examples								
Set weight	field		1.00	- Lucy	luo.	bu	1.15	1.10	
Set priors			S Linear	discriminan	t analysis res	ulte			X 14
			KK Linear	uisciinninan	· · · ·				06
Set costs					Discrim	ninant functio	ns		40
Set positive	e class value			A	С	В			02
					0.0054	0.0048			73
Learning	•								10
Stop analy	sis						_		97
Classificati	on	•					_		79
Test							_		86
							_		76
LIFT RO	C curve			_			_		16
							_		76
Error meas	urements	►					_		93
				_			_ ┣		74
Feature sel	ection	►					_		70
							_		82
Personnal	tests	•					_		57
	18	4					_		05
scriminant analus	19	4					_		86
	20	c		_			_		26
							_		82
									24
		-							78
							_		44
		-	Constant	-40.0487	-42.6622	-42.7323			23
									67
ted		-	0.42	0.54	4.00	4.44	2.55	2.05	
		۹.							4.91
	Set priors Set costs Set positive Learning Stop analy Classificati Test LIFT ROU Error meas Feature set	Learning Stop analysis Classification Test LIFT ROC curve Error measurements Feature selection Personnal tests scriminant analys nearDA 19 20 21 22 12 23 24 25 26	Set priors Set costs Set positive class value Learning Stop analysis Classification Test LIFT ROC curve Error measurements Feature selection Personnal tests Scriminant analyst nearDA ReadDA Second A Second A Sec	Set priors St. Linear Set costs Set positive class vule V1 Learning V2 V3 Stop analysis V3 V4 Classification V V7 LIFT ROC curve V8 V1 Feature selection V11 V1 Feature selection V12 V13 Personnal tests V14 V16 V17 V18 V17 21 B V23 C 23 C V20 V21 24 A Constant Constant 25 C 26 A 27 A -0.43 0.50 0.43 0.50	Set priors Set costs Set costs Set positive class value Learning A Stop analysis V1 Classification V Test V2 LIFT ROC curve V8 Error measurements V1 V10 1.9282 V11 1.9100 V12 1.91423 Stop analysis V1 LIFT ROC curve V8 Personnal tests V11 19 A 20 C 21 B 22 B 23 C 24 A 25 C 26 A 27 A -0.43 0.54	Set priors Step costs Discriminant analysis residuation Set costs A C Set positive class value A C Learning A C Stop analysis V1 0.0038 0.0054 Classification V 2.06041 0.4289 V3 1.0239 0.8771 V4 1.5208 1.2870 Classification V5 2.0835 1.7595 V6 2.6259 2.2604 V7 3.0398 2.8056 LIFT ROC curve V8 2.4858 2.4438 V9 1.9423 2.0855 Error measurements V10 1.9282 2.2103 V11 1.9100 2.3465 V10 1.9282 2.2103 V11 1.9100 2.3465 V11 1.9100 2.3465 V12 1.9149 2.2340 V11 1.9100 2.3465 V13 1.9370 2.2383 Personnal tests V16 2.5310	Set priors Set costs Discriminant analysis results Set costs A C B Set costs VI 0.0038 0.0054 0.0048 Learning VI 0.0038 0.0054 0.0048 Stop analysis VI 0.0038 0.0054 0.0048 V2 0.5041 0.4289 0.4740 V3 1.0239 0.8771 0.5618 V4 1.5208 1.2970 1.4100 Classification V V 2.0835 1.7595 1.9201 V6 2.6259 2.2504 2.4598 2.4438 2.6471 V9 1.9423 2.0855 2.2490 2.2012 V10 1.9282 2.2103 2.2970 V11 V11 1.9149 2.2840 2.2012 V11 1.9149 2.2840 2.2012 V13 1.9370 2.2383 2.0760 V14 2.4690 2.6217 2.4210 <t< td=""><td>Set priors Set costs Discriminant analysis results Set costs Discriminant functions Set positive class vulue A C B Learning V1 0.0034 0.0044 V2 0.6041 0.4269 0.4740 Stop analysis V3 1.0239 0.8771 0.9818 V4 1.5208 1.2970 1.4100 V5 2.0836 1.7895 1.9201 V6 2.6529 2.2504 2.4598 V7 3.0398 2.8056 3.0551 LIFT ROC curve V8 2.4858 2.4436 2.6471 V9 1.9423 2.0855 2.2490 V10 1.9822 2.2103 2.2970 V10 1.9109 2.4485 2.4471 V9 1.9423 2.0855 2.2490 V11 1.9100 2.4485 2.4471 V12 1.9149 2.840 2.012 V12 1.9149 2.8450 2.617 2.4210 <</td><td>Set priors St costs Discriminant analysis results Set costs Discriminant functions Set positive class vule A C B Learning 0.0038 0.0044 0.0048 V2 0.5041 0.4289 0.4740 Stop analysis V3 1.0239 0.8771 0.9618 V4 1.5208 1.2970 1.4100 V5 2.6855 2.7804 2.4958 V7 3.0398 2.8056 3.0551 LIFT ROC curve V8 2.4858 2.4438 2.6471 V9 1.9423 2.0855 2.2490 2.012 V11 1.9100 2.3485 2.4438 2.6471 V9 1.9423 2.0855 2.2490 2.012 V11 1.9100 2.3485 2.3477 V11 1.9100 2.3485 2.3487 V11 1.9100 2.4852 2.2810 V12 1.9149 2.2404 2.2012</td></t<>	Set priors Set costs Discriminant analysis results Set costs Discriminant functions Set positive class vulue A C B Learning V1 0.0034 0.0044 V2 0.6041 0.4269 0.4740 Stop analysis V3 1.0239 0.8771 0.9818 V4 1.5208 1.2970 1.4100 V5 2.0836 1.7895 1.9201 V6 2.6529 2.2504 2.4598 V7 3.0398 2.8056 3.0551 LIFT ROC curve V8 2.4858 2.4436 2.6471 V9 1.9423 2.0855 2.2490 V10 1.9822 2.2103 2.2970 V10 1.9109 2.4485 2.4471 V9 1.9423 2.0855 2.2490 V11 1.9100 2.4485 2.4471 V12 1.9149 2.840 2.012 V12 1.9149 2.8450 2.617 2.4210 <	Set priors St costs Discriminant analysis results Set costs Discriminant functions Set positive class vule A C B Learning 0.0038 0.0044 0.0048 V2 0.5041 0.4289 0.4740 Stop analysis V3 1.0239 0.8771 0.9618 V4 1.5208 1.2970 1.4100 V5 2.6855 2.7804 2.4958 V7 3.0398 2.8056 3.0551 LIFT ROC curve V8 2.4858 2.4438 2.6471 V9 1.9423 2.0855 2.2490 2.012 V11 1.9100 2.3485 2.4438 2.6471 V9 1.9423 2.0855 2.2490 2.012 V11 1.9100 2.3485 2.3477 V11 1.9100 2.3485 2.3487 V11 1.9100 2.4852 2.2810 V12 1.9149 2.2404 2.2012

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

Label	А	В	С
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
	SIPIN	A	

Linear D)iscriminan	t Function f	or ONDE
Variable	Α	В	С
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...).

Sipina Research Version 3.1	10	x
File Edit Data Statistics	Induction method Analysis View Window Help	
🖸 🕂 🙀 🛤	Standard algorithm (1)	
Attribute selection	Select an induction method (2)	
🖻 🗊 Predictive attributes	Induction Graph Rule Induction Neural network Discriminant analysis Decision list Other	Ξ
	Linear discriminant analysis -0.14	F
	Multithreaded Linear discriminant analysis (memory parsimonious) (3) -0.06	;
Learning method	2.40	_
MethodName=Linear discriminant MethodClassName=TLinearDA	0.02	_
Hdl=11	1.73	_
Priors=2	LDA - Mutliple threads	_
Costs=1	Priors	_
	C ∐ser defined	-
J	C Same of all classes	_
Examples selection	Multithread Class Distribution 0.76	_
500000 examples selected	·····································	_
0 examples idle	(5) V DK 3.76	
Linear discriminant analysis	Exec.Time : 1295 ms	

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.

Induction method Analysis View	v Window	Help						
🖰 🕂 🎬 🕒				_				
Attribute selection ⊡r∰ Predictive attributes	👗 Lear	🛛 🔊 Linea	ar discriminan	t a ysis res	👝 🖸		l	
		-	Discriminant functions				V5	
	1		A	C	в		-0.35	
	2	V1	0.00377	0.00541	0.00476		0.07	
Learning method	3	V2	0.50408	0.42693	0.47400		0.47	
MethodName=Linear discriminant analys	4	V3	1.02392	0.87710	0.96181		1.76	
MethodClassName=TLinearDAThread Hdl=11	5	V4	1.52079	1.29700	1.41004		0.99	
Priors=2	6	V5	2.06354	1.75948	1.92008		-0.37	-0.37
Costs=1	7	V6	2.52591	2.25037	2.45984		2.10	
	8	V7	3.03977	2.80563	3.05506		2.18	
Examples selection 500000 examples selected 0 examples idle	9	V8	2.48582	2.44356	2.64713		2.27	
	10 19	1.94234	2.08549	2.24896		1.18		
	11	V10	1.92816	2.21027	2.29702	-	0.36	
	12		-0.11	2.31	0.52	-0.17	2.05	

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.

Gestionnaire des tâches de Windows	-	-	
Fichier Options Affichage ?			
Applications Processus Services Performance	Mise en rése	ريلي	tilisateurs
Nom de l'image	Nom d'u	\checkmark	Description
Sipina_Research.exe *32	Maison	75	Sipina_Research.exe
	Maison	-00	R STATE OF THE REAL PROPERTY O
	Maison	00	
	Maison	00	North Control of Contr
	Maison	00	
	Maison	00	
	Maison	00 🤇	
	Maison	00	
	Maison	00	and the second
	Maison	00	
	Maison	00	-
-			P.
Afficher les processus de tous les utilisate	eurs		Arrêter le processus
Processus : 67 UC utilisée : 74% Mén	noire physiq	ue : 30	%

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 = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 121 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 120 descriptors (100 additional descriptors); WAVE2M with n = 500,000 instances and p = 120 descriptors (100 additional descriptors); WAVE2M with n = 500,000 descriptors (100 additional descriptors); WAVE2M with n = 500,000 descriptors (100 additional descriptors); WAVE2M with n = 500,000 descriptors); WAVE2M

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 COVTYPE⁵ dataset with K = 7 classes, but rather unbalanced (2 classes of the target variable concentrated a large part of the observations); and MIT FACE IMAGE⁶ dataset, with K = 2, but with very unbalanced classes.

In the table below, we show the execution time in seconds⁷. "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	К	n	р	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.

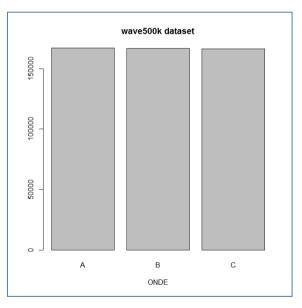


Figure 1 – Class values distribution for "wave500K"

⁵ <u>http://archive.ics.uci.edu/ml/datasets/Covertype</u>

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.

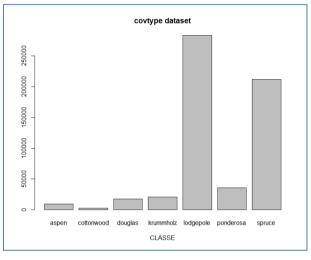


Figure 2 – Class values distribution for "covtype"

• 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).

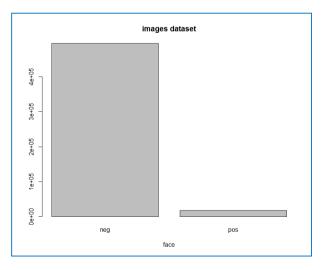


Figure 3 – Class value distribution for "mit face images"

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 dataset⁸.

<u>File D</u> iagram Compo	onent <u>W</u> indow <u>H</u> elp									_ đ	5
) 📽 🔚 🏭											
	Default title		MANOVA								1
Dataset (wave500k.	txt)		Stat	Value	p-value						
🖻 🚰 Define status 1			Wilks' Lambda	0.2922							
Supervised L	Supervised Learning 1 (Linear discriminant analysi			615194.3996			Λ\				
			Bartlett C(42) Rao F(42, 99995				A)				
			Ra0 P(42, 99995	4) 20236-254/	0.00	00					
			LDA Summ	ary							
			Classification functions					Statistical Evaluation			
			Attribute	A	с	В	Wilks L.	Partial L.	F(2,499977)	p-value	l
			V1	0.003766	0.005411	0.004762	0.292170	1.000000	0.06341	0.938563	3
			V2	0.504083	0.426932	0.474002	0.292331	0.999447	138.37781	0.000000	5
			V3	1.023918	0.877096	0.961812	0.292765	0.997967	509.33504	0.000000	5
			V4	1.520789	1.296996	1.410041	P 35 B	0.995135	1222.07484	0.00000	5
			V5	2.063536	1.759484	1.920077	0.2949_0	0.990574	2378.89153	0.000000	5
	III	•	V6	2.525914	2.250373	2.459838	0.294643	0.991605	2116.53458	0.000000)
				Components							
Data visualization	Statistics	Nonpara	metric statistics	Instance se	lection	Feature	construction	Featu	ire selection		
Regression	Factorial analysis		PLS	Clusteri	ng	Spv	Spv learning Met		-spv learning		
pv learning assessment	Scoring	A	ssociation								
Binary logistic regressio			Ats CS-MC4			ecision List		🔣 Linea	r discriminant	analysis	
§ BVM	🧟, C-RT		🗽 c-svc		- <mark>na</mark> ∎D				eg TRIRLS		
s C4.5	As CS-CRT		🐹 CVM		[∴ К	NN		🔉 Multi	ayer perceptr	ən	ł

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 <u>SAS</u> 9.3 (proc discrim), <u>R</u> 3.0.0 (64 bits) and <u>Revolution R</u> <u>Community</u> 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 <u>http://data-mining-tutorials.blogspot.fr/2012/11/linear-discriminant-analysis-tools.html</u> 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 **Ida()** 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)</pre>
```

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

Dataset	К	n	р	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 induction⁹) than the others (e.g. support vector machine¹⁰, 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 paper¹¹.
- 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 <u>http://data-mining-tutorials.blogspot.fr/2009/07/implementing-svm-on-large-dataset.html</u>, 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, <u>10th printing</u>, 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 <u>http://cran.r-project.org/web/views/HighPerformanceComputing.html</u>