

1 Topic

Comparison of the calculation times of various tools during the logistic regression analysis.

The programming of fast and reliable tools is a constant challenge for a computer scientist. In the data mining context, this leads to a better capacity to handle large datasets. When we build the final model that we want to deploy, the quickness is not really important. But in the exploratory phase where we search the best model, it is decisive. It improves our chance to obtain the best model simply because we can try more configurations.

I have tried many solutions to improve the calculation times of the logistic regression. In fact, I think the performance rests heavily on the optimization algorithm used. The source code of Tanagra shows that I have greatly hesitated. Some studies have helped me about the right choice¹.

Several tools propose the logistic regression. It is interesting to compare their calculation times and memory occupation. I have already studied this kind of comparison in the past². The novelty here is that I use a new operating system (64 bit version of Windows 7), and some tools are especially intended for this system. The calculating capabilities are greatly improved for these tools. For this reason, I have increased the dataset size. Moreover, to make more difficult the variable selection process, I added predictive attributes that are correlated to the original descriptors, but not to the class attribute. They have not to be selected in the final model.

In this paper, in addition to **Tanagra 1.4.14** (32 bit), we use **R 2.13.2** (64 bit), **Knime 2.4.2** (64 bit), **Orange 2.0b** (build 15 oct2011, 32 bit) and **Weka 3.7.5** (64 bit).

2 Dataset

Choosing the appropriate dataset for an experiment is always difficult. The specific properties of the dataset must not interfere with the tools characteristics. The risk is to obtain biased results. This is one of the reasons why I use often the same datasets of which I know the specificities.

“**Waveform**³” is one of my favorite datasets, partly because we can generate as instances as we want. We can also add descriptors which are generated randomly, or descriptors which are correlated to the existing ones. So, we can study the tools behavior on a potentially infinite dataset.

We have transformed the waveform dataset in a binary problem by dropping the instances corresponding to the last class value in this tutorial. Indeed, the standard logistic regression can handle only binary problems. Below, we describe the R program we used to generate the dataset.

2.1 Waveform

We use the R source code available online to generate the original waveform dataset⁴. It generates a dataset with "n" instances, the class attribute with 3 values, and the 21 original descriptors.

¹ T.P. Minka, « [A comparison of numerical optimizers for logistic regression](#) », 2007.

² Tanagra tutorial, « [Logistic regression – Software comparison](#) », december 2008.

³ <http://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+%28Version+1%29>

⁴ T. Hastie, R. Tibshirani, J. Friedman, « [The elements of statistical learning](#) », Springer, 2009.

```
#from http://www-stat.stanford.edu/~tibs/ElemStatLearn/data.html
waveform <- function(n)
{
  class <- as.numeric(cut(runif(n), c(0, 1/3, 2/3, 1)))
  h <- function(xoff)
    pmax(6 - abs(seq(21) - 11 + xoff), 0)
  x <- rbind(h(0), h(-4), h(4))
  x1 <- x[c(1, 1, 2), ][class, ]
  x2 <- x[c(2, 3, 3), ][class, ]
  u <- runif(n)
  data.frame(x = I(u * x1 + (1 - u) * x2 + rnorm(n * 21)), y = class)
}
```

2.2 “Binary” waveform

To transform the waveform problem in a binary one, we use the following strategy: we generate more instances than needed first, then we remove the instances corresponding to the third class value. If we obtain more rows than needed, we truncate the dataset. It is not a problem because the order of the instances is not correlated to the class value.

```
#modify waveform in a binary problem by removing
#the instances for the third class value
waveform.binary <- function(n){
  tmp <- waveform(trunc(1.75*n))
  output <- subset(tmp, tmp$y != 3)
  if (nrow(output) > n){
    output <- output[1:n,]
  }
  output$y <- factor(output$y)
  levels(output$y) <- c("A", "B")
  return(output)
}
```

2.3 Adding the “random” descriptors

We add random descriptors to the original dataset. They should not be selected during the feature selection process. To obtain credible descriptors, they are created according a Gaussian distribution using the characteristics of the 21 original attributes (mean, standard deviation).

```
#add K random attributes
add.rnd <- function(wave.data, K = 1){
  n <- nrow(wave.data)
  new.wave <- wave.data
  for (k in 1:K){
    colref <- trunc(runif(1,min=1,max=22))
    newcol <- rnorm(n,mean=mean(wave.data$x[,colref]),sd=sd(wave.data$x[,colref]))
    new.wave <- cbind(new.wave,newcol)
    names(new.wave)[ncol(new.wave)] <- paste("rnd",k,sep="_")
  }
  return(new.wave)
}
```

2.4 Adding the “correlated” descriptors

To boost the difficulty, we add correlated attributes to the dataset i.e. we select randomly one the original attributes and we add a noise. When the noise is weak, the correlation between the attributes is strong. We observe that these new descriptors are generated in such a way that they are correlated to the original descriptors, but not to the class attribute. They must be discarded during the variable selection process.

```
#add L correlated attributes
add.correlated <- function(wave.data,L = 1, noise=1){
  n <- nrow(wave.data)
  new.wave <- wave.data
  for (l in 1:L){
    colref <- trunc(runif(1,min=1,max=22))
    newcol <- wave.data$x[,colref] + rnorm(n,0,sd=sd(wave.data$x[,colref])*noise)
    new.wave <- cbind(new.wave,newcol)
    names(new.wave)[ncol(new.wave)] <- paste("cor",l,sep="_")
  }
  return(new.wave)
}
```

2.5 Main program for generating the dataset

The following program is used for generating the dataset. We set the following settings: n = 300,000 instances, 21 original descriptors (V), the binary class attribute (Y), 50 descriptors generated randomly (rnd), 50 correlated descriptors (cor). The values are rounded to the third decimal.

Of course, the reader can generate a dataset with other characteristics (n, rnd, cor) according to their context and their goal.

```
#function for rounding numeric columns
myround <- function(x){
  if (is.factor(x)==T){
    return(x)
  } else
  {
    return(round(x,3))
  }
}

#generate n instances with K rnd attributes and L correlated attributes
generate.binary <- function(n,K=1,L=1,noise=1){
  tmp <- waveform.binary(n)
  tmp <- add.rnd(tmp,K)
  tmp <- add.correlated(tmp,L,noise)
  output <- cbind(as.data.frame(matrix(tmp$x,nrow(tmp),21)),subset(tmp,select=-x))
  output <- as.data.frame(lapply(output,myround))
  return(output)
}
```

```
#generate and save a dataset
dataset.size <- 300000 #number of instances
nb.rnd <- 50 #number of random variables
nb.cor <- 50 #number of correlated variables
noise.level <- 1 #noise for correlated variables
data.wave <- generate.binary(dataset.size,nb.rnd,nb.cor,noise.level)
summary(data.wave)

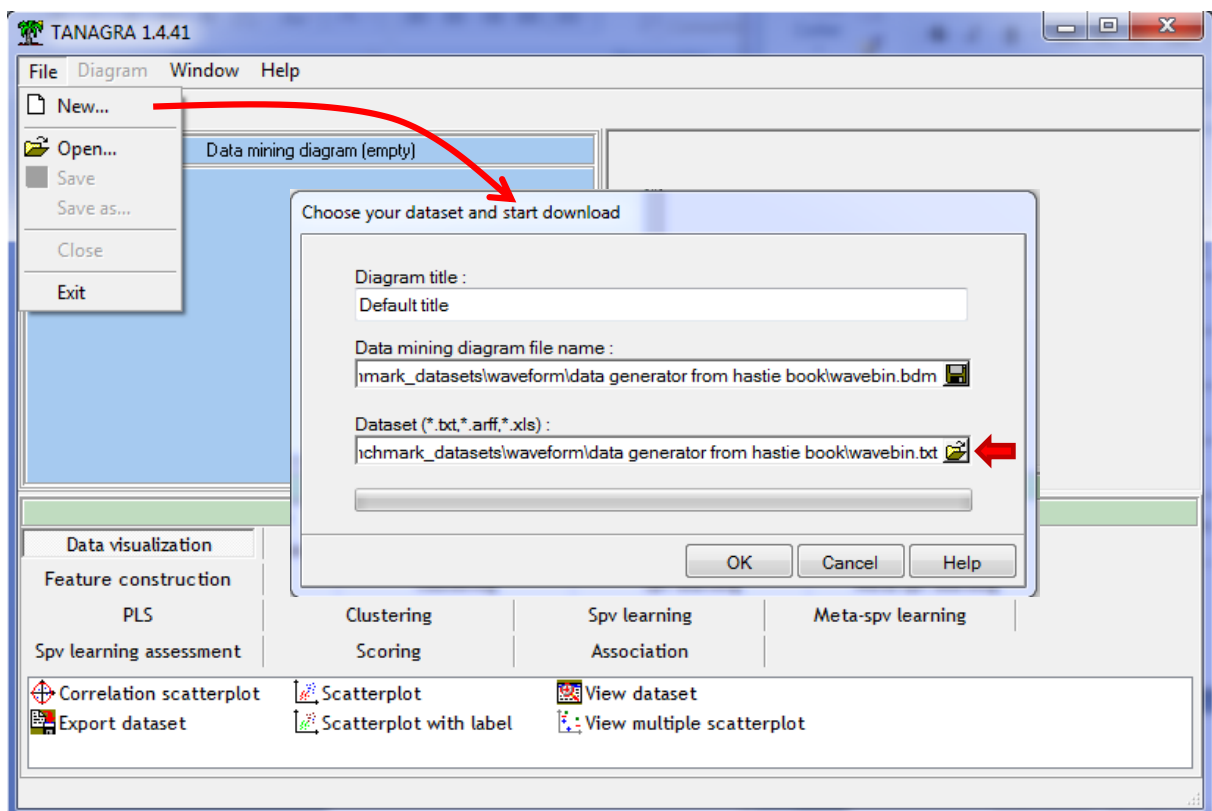
#writing the data.frame into a file (tab delimited file format)
write.table(data.wave,file="wavebin.txt",quote=F,sep="\t",row.names=F)
```

3 Logistic regression with Tanagra

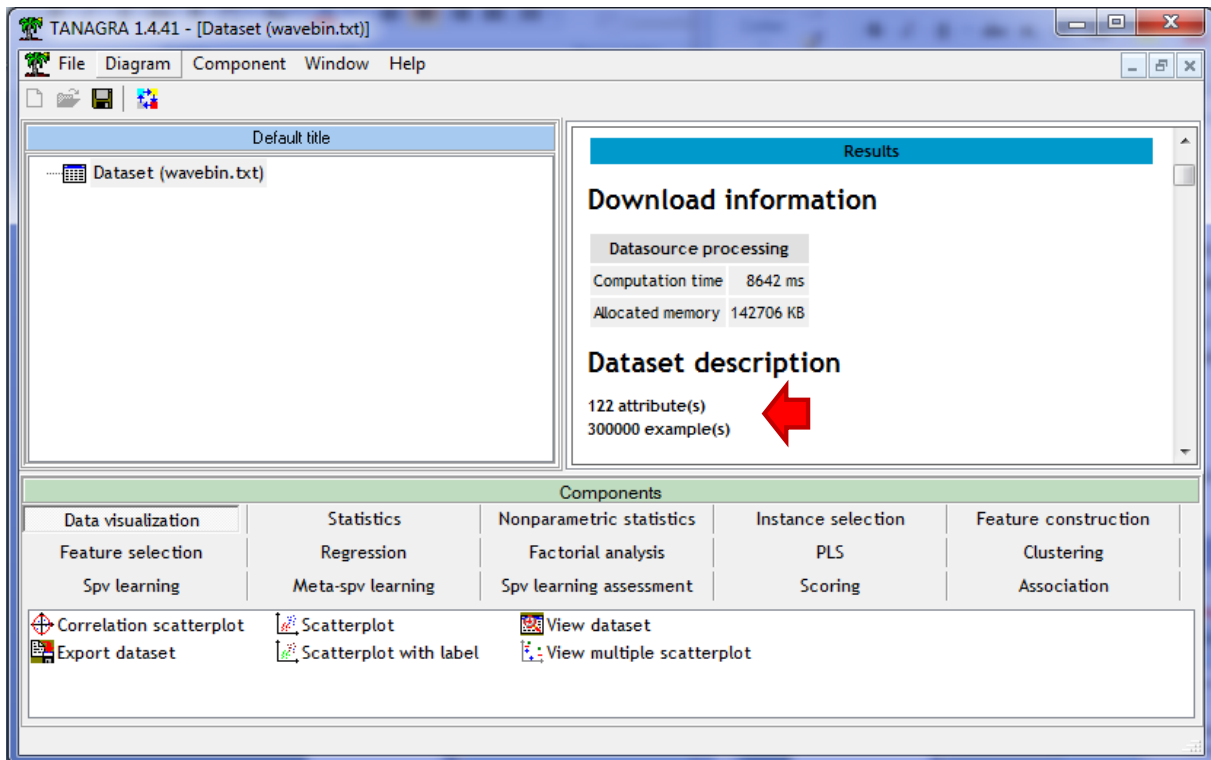
We use the [wavebin.txt](#) data file for our experiment. Because some tools start automatically the learning process when we define the diagram, we use a small dataset with $n = 300$ instances for the screenshots ([wavebin_small.txt](#)).

3.1 Tanagra

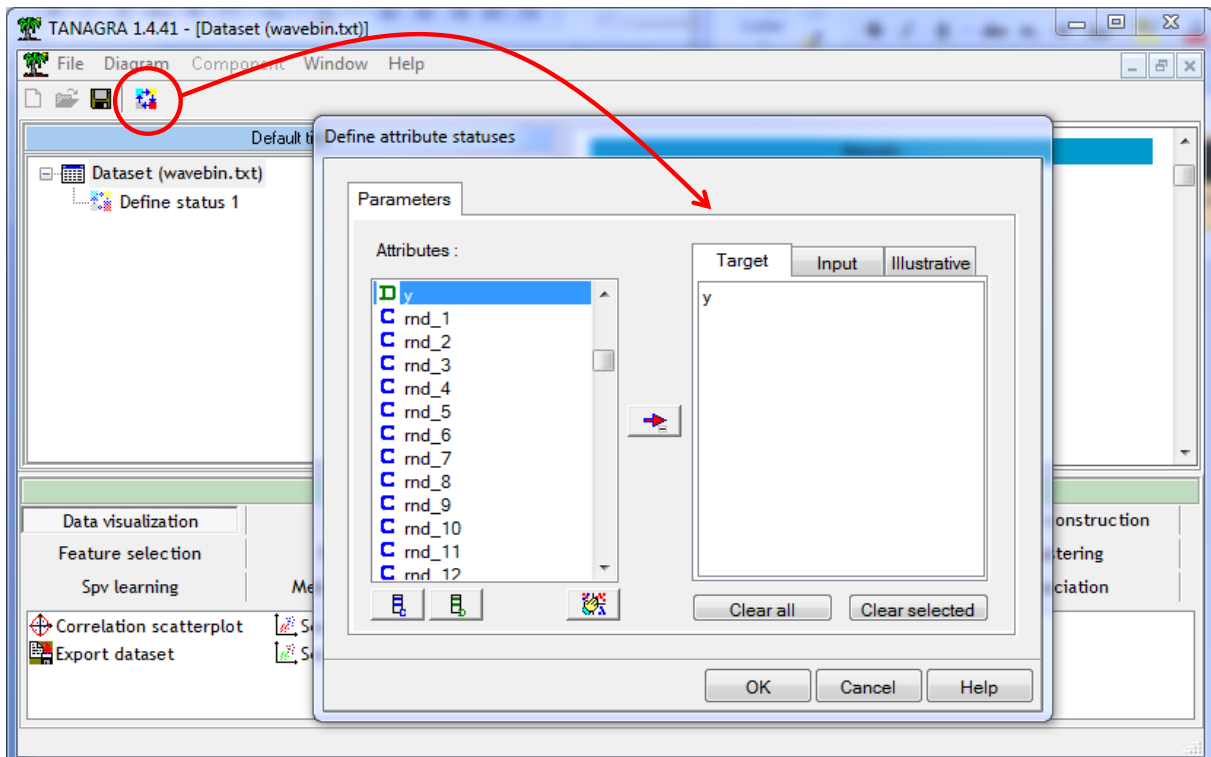
We click on the FILE / NEW menu to create a new diagram. We import the dataset.



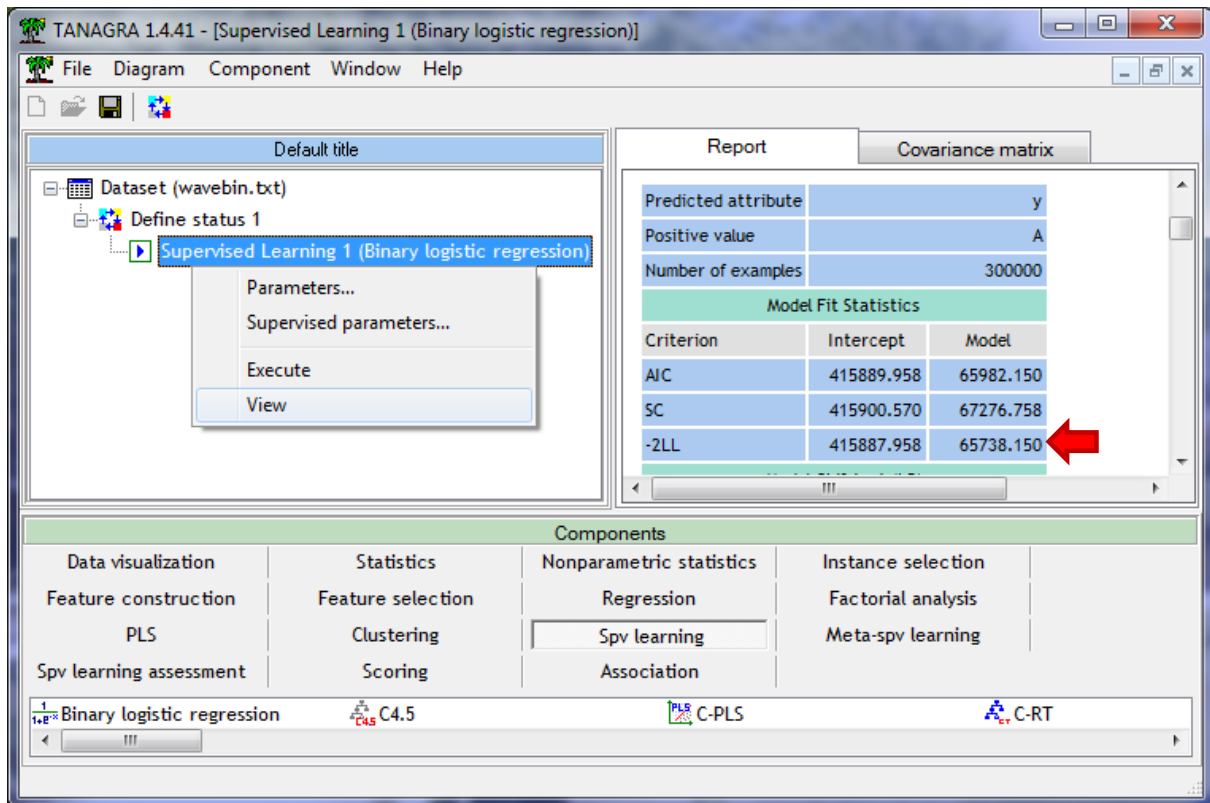
We check that the values are correctly imported. We have 300,000 rows and 122 columns.



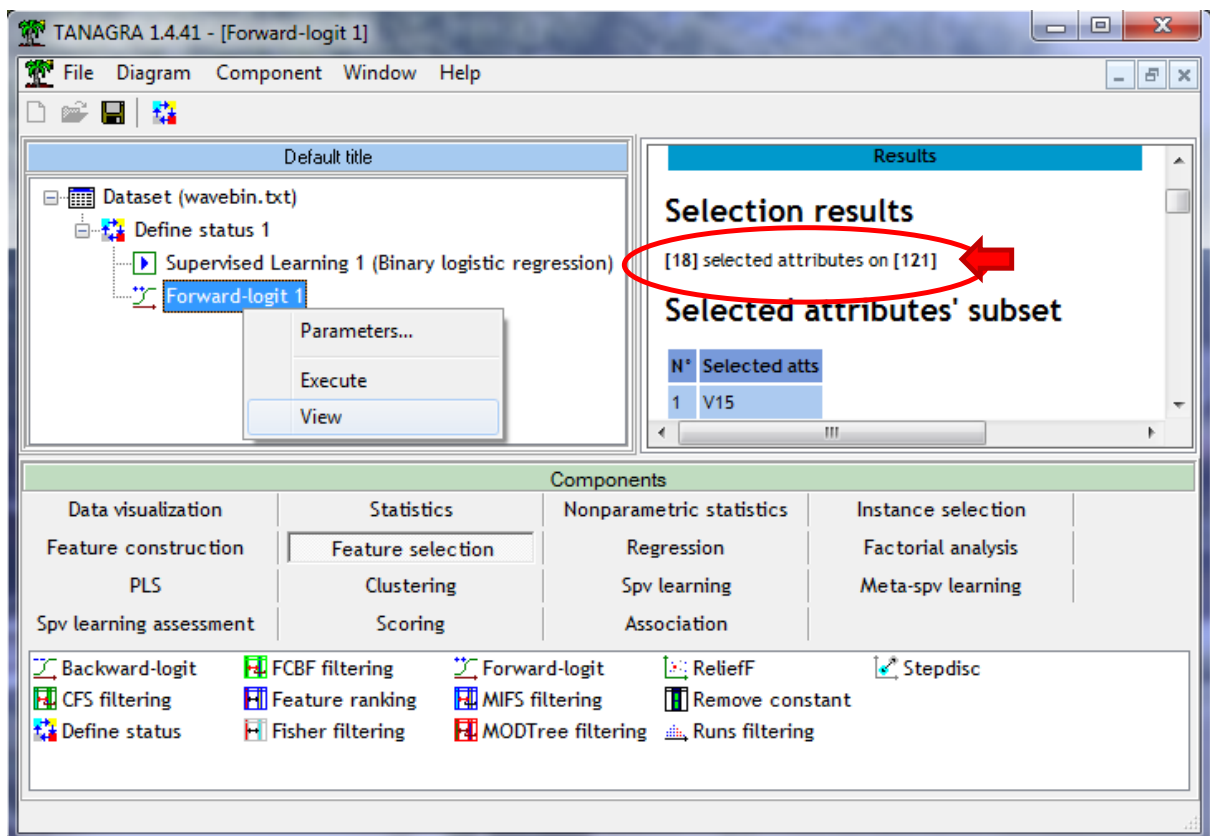
We specify the **target** attribute (Y) and the **input** ones (the others).



We add the BINARY LOGISTIC REGRESSION (SPV LEARNING tab) into the diagram. We launch the process by clicking on the VIEW contextual menu. The obtained deviance is **D = 65738.10**. This is the reference value that we use to check the optimization quality of the various tools.



For the variable selection, we add the FORWARD LOGIT (FEATURE SELECTION tab) component to the diagram. We use the default settings.



3.2 R software

We use the following program under R. The calculation times are measured with the `system.time(.)` command.

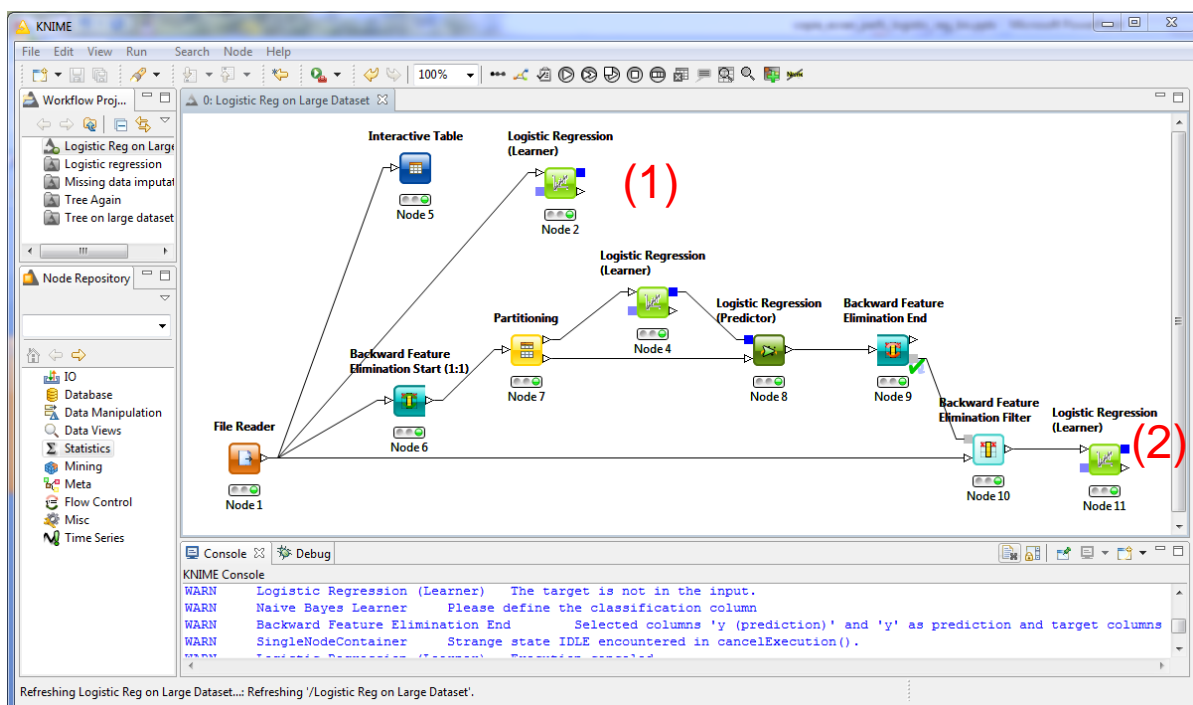
```
#data importation
system.time(wave
read.table(file="wavebin.txt",sep="\t",dec=".",header=T))

#creating the model
system.time(model <- glm(y ~ ., data = wave, family=binomial))
print(model)

#variable selection
library(MASS)
model.default <- glm(y ~ 1, data = wave, family=binomial)
duree <- system.time(model.forward <-
stepAIC(model.default,scope=list(lower=as.formula(model.default),upper=as.f
ormula(model)),direction="forward",k=log(nrow(wave))))
print(model.forward)
```

3.3 Knime

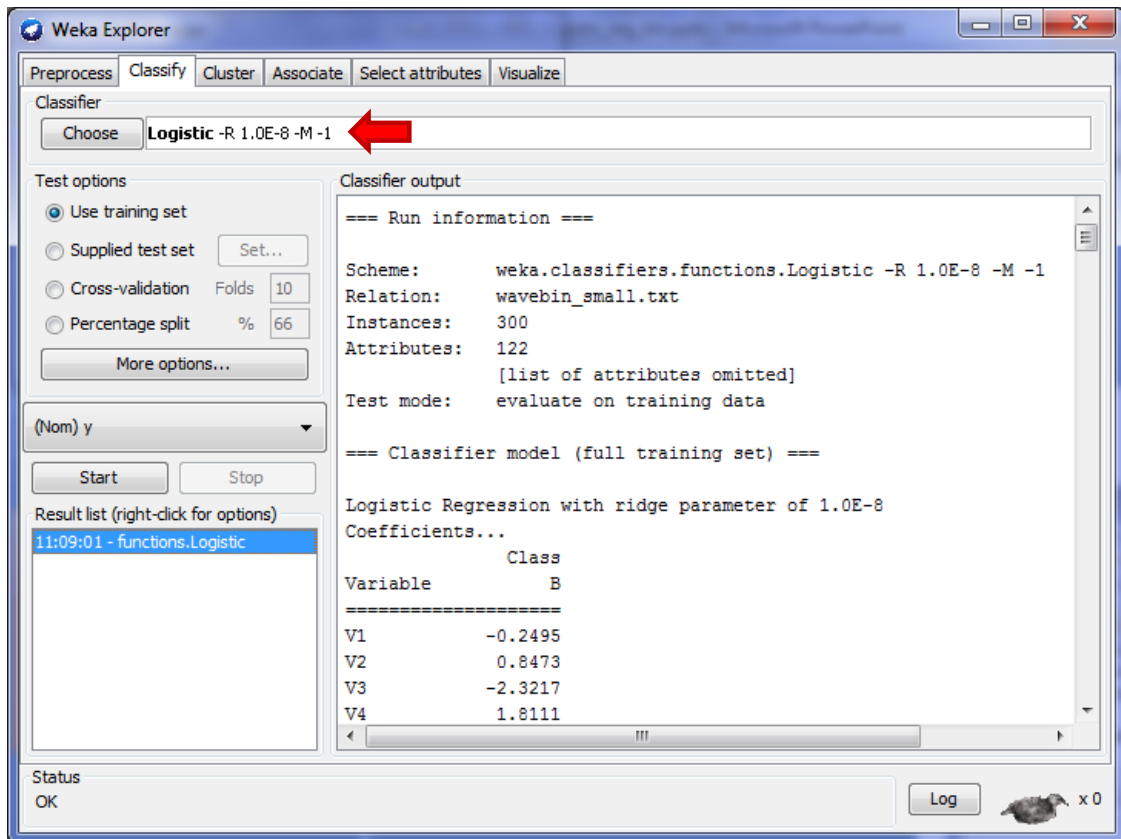
We define the following project under Knime. We have the learning process on the dataset (1), and the variable selection process using the wrapper approach (2)⁵.



3.4 Weka

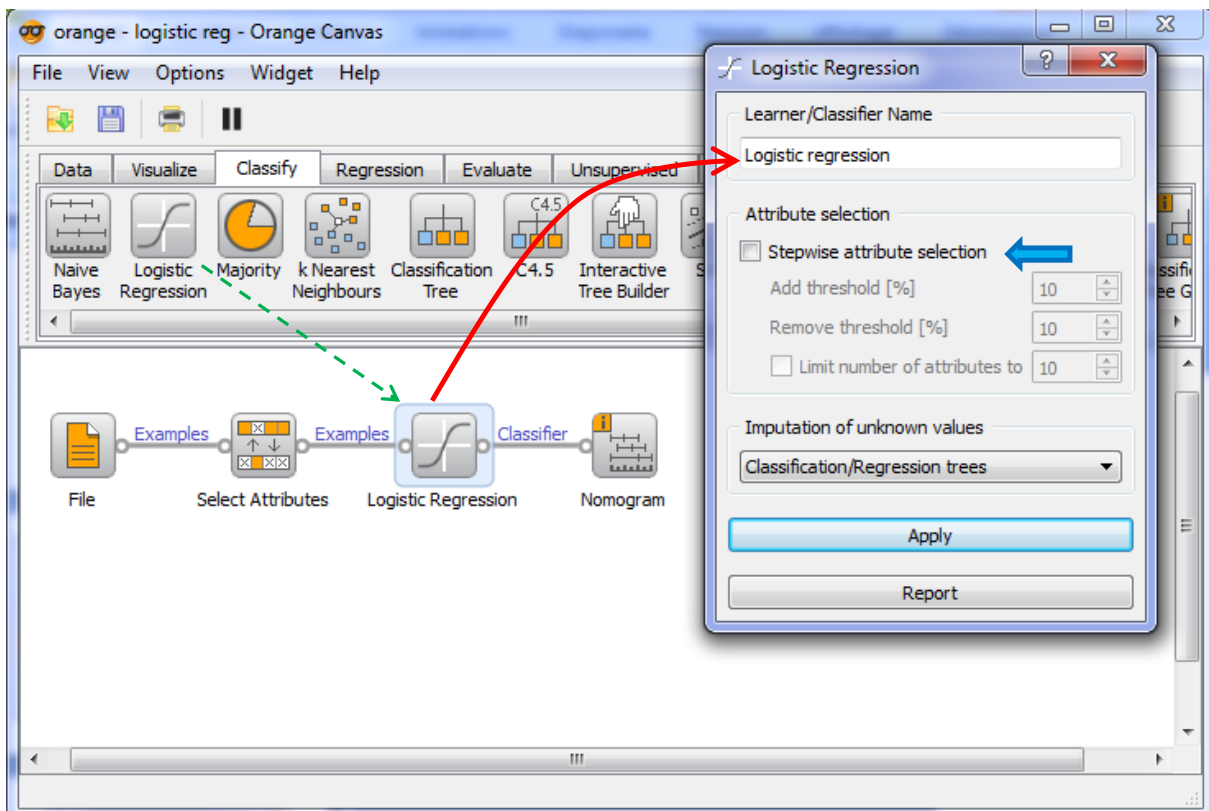
We use the Explorer mode under Weka. After the data importation, we launch the Logistic procedure with the following settings.

⁵ Tanagra tutorial, « [Wrapper for feature selection \(continuation\)](#) », april 2010.



3.5 Orange

We use the following schema under Orange. We note that the variable selection procedure is incorporated into the Logistic Regression component.



3.6 Performances comparison

We detail the results into the table below. We note that all tools obtain the same deviance $D = 65738.15$. The quality of the model is the same whatever the tool used. This is a positive result for all the tools.

Tool (x bit)	Calculation time (seconds)		Memory occupation (GB)		
	Importation	Learning phase	After the data importation	Max during the learning process	Gap
Tanagra 1.4.41 (32)	9 ⁶	74	0.15	0.37	0.22
R 2.13.2 (64)	51	171	0.57	2.29	1.72
Knime 2.4.2 (64)	34	192	2.95	3.84	0.89
Weka 3.5.7 (64)	63	300	2.1	2.41	0.31
Orange 2.0b (32)	151	-	1.27	-	-

3.6.1 Calculation time

We use the values provided by the tools if they exist. Otherwise, we use a chronometer. Of course, we obtain approximate values, but when the gap between the performances is high, a precise value is not important.

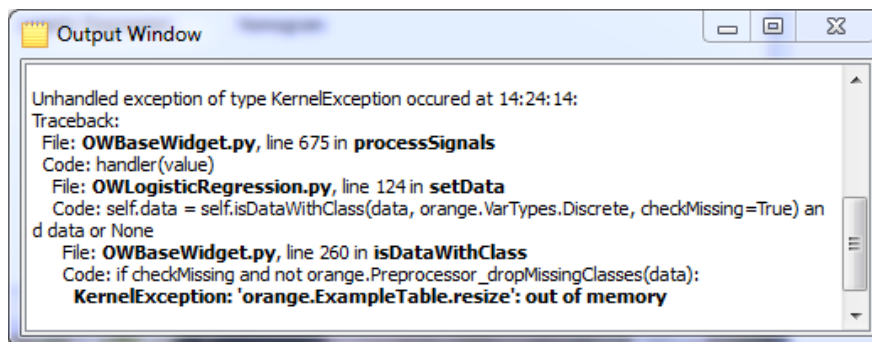
Tanagra is really faster compared with the other tools. As I said above, I took a lot of time to improve the program. But I think that the other reason is the optimization algorithm used. The Minka's work referenced above was a considerable help.

R and **Knime** are also very fast. They work in a 64 bit mode.

Knime can store the dataset on the disc when the memory occupation is too high (<http://tech.knime.org/faq> - see « *Memory Policy* »). This feature is useful when we handle a large dataset. But on the other hand, the calculation time becomes slower. To make the comparison possible, we use the option "Keep all in memory" in our experiment. If we use the default option, the calculation time for the learning phase is twofold (about 7 minutes).

On our small dataset (300 instances), **Orange** works properly. When we select the large dataset (300,000 instances), the data file is imported, but the following error message appears when the logistic regression begins.

⁶ For the **1.4.42 and posterior versions**, the importation time is higher (about 11 seconds) because **Tanagra checks now the missing values**.



```

Output Window
Unhandled exception of type KeyboardInterrupt occurred at 14:24:14:
Traceback:
File: OWBaseWidget.py, line 675 in processSignals
Code: handler(value)
File: OWLogisticRegression.py, line 124 in setData
Code: self.data = self.isDataWithClass(data, orange.VarTypes.Discrete, checkMissing=True) and
data or None
File: OWBaseWidget.py, line 260 in isDataWithClass
Code: if checkMissing and not orange.Preprocessor_dropMissingClasses(data):
KernelException: 'orange.ExampleTable.resize': out of memory

```

I thought first that that we can overcome this problem by modifying the settings (as for Java JRE). But, the problem seems more difficult (<http://orange.biolab.si/forum/viewtopic.php?f=4&t=1369>).

3.6.2 Memory occupation

We measure the memory occupation by using the Windows task manager. We keep the maximum value reached during the learning phase. This is rather a handcrafted process, but this is the most reliable. Indeed, some tools removed the unused structure after the building of the model, the memory occupation measured after the learning phase is not really accurate.

The "gap" column computes the gap between the memory occupation before the learning and the max reached during the model construction. An interesting idea for instance is to measure the changes when we modify the number of instances and/or the number of descriptors.

The global memory occupation gives the ability of the tool to handle large database... when they handle all the instances into main memory. About Knime for instance, when we use the default option (the tables are written to disc), the used memory is really small (about 0.38 GB). It reinforces its ability to operate on large databases.

Last, about Tanagra, the memory occupation seems really small because it stores the values in single precision. We do not need a high performance for the storage of the values. Conversely, all the calculations are made in double precision to obtain as accurate results as the other tools.

4 Variable selection

The tools used different algorithms for the variable selection. Thus, the calculation times are not comparable in the absolute. They rest on the number of logistic regression learning performed during the search i.e. the number of times where we try to optimize the log-likelihood. This operation is the most time-consuming.

Let "p" the number of candidate variables.

Tanagra uses the Score test for the forward approach, and the Wald test for the backward approach. The search is performed in a linear time $O(p)$ i.e. in the worst case, we perform "p" logistic regression learning. For the forward approach, we begin with the simplest models. The algorithm is faster.

Although the simplicity of the method, we observe that none of the irrelevant descriptors ("rnd" or "cor") are incorporated into the final model with Tanagra.

R avec stepAIC optimizes the Akaike (AIC) criterion. According the forward search, we try all the regression with 1 predictor. We perform "p" logistic regression learning process. Then, we select the

best one. We try to add a second variable. So, we perform "p-1" regressions. The algorithm is quadratic $O(p^2)$. The calculation time is heavily impacted.

Knime provides the wrapper approach with the backward search strategy. In our project, we optimize the error rate computed on a separate test set. The algorithm is also quadratic, but because we start with the complete model, the calculation time becomes prohibitive. On our small dataset (300 instances), it works fine. After a long time, we cut off the calculations on the large dataset (300,000 instances). Obviously, this kind of approach is only tractable with a very fast learning algorithm such as naive bayes classifier⁷.

Weka, as Knime, does not incorporate a variable selection procedure especially intended to the logistic regression. Among the possible approaches, we can use the CFS filtering algorithm⁸. But, it is based on a criterion (the correlation) which is not directly related to the logistic regression characteristics. This is the reason for which we do not include this procedure in our experiment.

Orange incorporates the bidirectional variable selection approach (stepwise) in the logistic regression. It works fine on our small dataset. On the large dataset, the calculation is not possible.

Finally, we report here the results for Tanagra and R (stepAIC).

Tool (Approach)	Selected descriptors			Calculation time
	# variables	Of which « rnd »	Of which « cor »	
Tanagra (Forward)	18	0	0	9' 30''
R (Forward, StepAIC)	18	0	0	4h 54' 00''

As we said above, the calculation time is not really relevant here because the tools do not rest on identical algorithms. We note however that they exclude the irrelevant descriptors "rnd" and "cor". Especially for the second type of descriptors, this is a really good result.

5 Conclusion

The reader can adjust the dataset characteristics (more or less instances and /or descriptors). The conclusions will be more relevant according to its context.

About the performances, Tanagra seems very fast compared with the other tools. The main reason is that I really worked on the optimization of this method (like for decision tree algorithm)⁹. The results are less outstanding for other approaches such as SVM¹⁰ (Support vector machine). There is still work to do.... This is also good news.

⁷ Tanagra tutorial, « [Wrapper fo feature selection \(continued\)](#) », april 2010.

⁸ Tanagra tutorial, « [Filter methods for feature selection](#) », october 2010.

⁹ Tanagra tutorial, « [Decision tree and large dataset \(continuation\)](#) », october 2011.

¹⁰ Tanagra tutorial, « [Implementing SVM on large dataset](#) », july 2010.