An R package to select continous variables for multiclass classification with a stochastic wrapper method

Kim-Anh Lê Cao, Patrick P. Chabrier

Journal of Statistical Software

November 2008, Volume 28, Issue 9. *http://www.jstatsoft.org/*

ofw: An R Package to Select Continuous Variables for Multiclass Classification with a Stochastic Wrapper Method

Kim-Anh Lê Cao
Université de Toulouse

Patrick Chabrier
Institut National
de la Recherche Agronomique

Abstract

When dealing with high dimensional and low sample size data, feature selection is often needed to help reduce the dimension of the variable space while optimizing the classification task. Few tools exist for selecting variables in such data sets, especially when classes are numerous (> 2). We have developed **ofw**, an R package that implements, in the context of classification, the meta algorithm "optimal feature weighting". We focus on microarray data, although the method can be applied to any $p \gg n$ problems with continuous variables. The aim is to select relevant variables and to numerically evaluate the resulting variable selection. Two versions are proposed with the application of supervised multiclass classifiers such as classification and regression trees and support vector machines. Furthermore, a weighted approach can be chosen to deal with unbalanced multiclasses, a common characteristic in microarray data sets.

Keywords: feature selection, stochastic algorithm, classification, unbalanced multiclass, microarray.

1. Introduction

Performing a feature selection algorithm has several important applications in high dimensional data sets. For example with microarray data, it is sensible to use a dimensional reduction technique, either to identify genes that contribute the most for the biological outcome (e.g., cancerous vs. normal cells) and to determine in which way they interact to determine the outcome, or to predict the outcome when a new observation (or sample) is presented. Such a method would provide practical aspects with machine learning methods: it avoids the "curse of dimensionality" that leads to overfitting when the number of variables is too large.

There are two ways of selecting features. Either explicitly (filter methods) or implicitly (wrapper methods). The filter methods measure the relevance of a feature at a time by performing statistical tests (e.g., t test, F test) and ordering the p values. This type of approach is robust against overfitting and is fast to compute. However, it usually disregards the interactions between the features as it tests one variable at a time. Chen *et al.* (2003) compared four filter methods and also reached this conclusion.

The wrapper methods measure the usefulness of a feature subset by searching the space of all possible feature subsets. The search can be performed either with heuristic or stochastic search. The main disadvantages of these methods are their tendency to overfit and when dealing with numerous variables, an exhaustive search is computationally impossible. However, the resulting selection takes into account the interactions between variables and might highlight useful information on the experiment. Despite this latter property, wrapper methods are still not widely applied in microarray data. Comparisons of random forests (Breiman 2001), recursive feature elimination (Guyon *et al.* 2002), l_0 norm support vector machine (Weston *et al.* 2003) and biological interpretation of the resulting gene selections is given in Lê Cao *et al.* (2007b).

In the package **ofw**, we implement the wrapper method "optimal feature weighting" (OFW) adapted from Gadat and Younes (2007) that numerically quantifies the classification efficiency of each variable with a probability weight, by using stochastic approximations. This meta algorithm can be applied to any classifier. Therefore, the classifiers SVM (support vector machines, Vapnik 1999) and CART (classification and regression trees, Breiman *et al.* 1984) have been implemented so as to select an optimal subset of dicriminative variables. Few wrapper methods have been proposed yet to deal with multiclass data sets (Li *et al.* 2004; Chen *et al.* 2003; Yeung and Burmgarner 2003), especially when the classes are unbalanced (Chen *et al.* 2004). Our function `ofw()` proposes a weighting approach to deal with this common characteristic in microarrays. The package **ofw** is freely available as an R package (R Development Core Team 2008) under the GPL license. The package can be downloaded from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=ofw`.

Furthermore, like any wrapper methods, **ofw** requires heavy computations, especially when the number of variables is large. In this package, some of the computation time has been reduced by implementing some C functions and by proposing parallel programming during the learning step.

Finally, we propose to perform the e.632+ bootstrap method (Efron and Tibshirani 1997) to estimate the classification error rate on bootstrap samples and to evaluate the different variants of OFW and the resulting gene selections.

The general principle of the OFW algorithm is first presented. We then detail how to use **ofw** by applying the main functions on one microarray data set that is available in the package.

2. Optimal feature weighting model

2.1. Principle

OFW (Gadat and Younes 2007) is a meta algorithm that can treat several classification problems with a feature selection task. Any classifier can be applied, and Lê Cao *et al.* (2007a) implemented OFW with CART and SVM for multiclass classification (see also Lê Cao *et al.*

2007b for binary case).

We assume that the n examples (or cases) are described by p attributes (or variables) and labelled with their target class (e.g., $\{0, 1\}$ in binary problems).

Given a probability weight vector $\mathbb{P}$ on all p variables, the key idea of OFW is to learn $\mathbb{P}$ such that it fits the classification efficiency of each variable in the given problem. In short, important weights will be given to variables with a high discriminative power, and low or zero weights to non relevant variables in the classification task.

For that purpose, the algorithm adopts a wrapper technique, by drawing a small variable subset ω at a time, by measuring the relevance of this subset with the computation of the classification error rate of a given classifier, and then by updating the probability weights $\mathbb{P}$ according to the discriminative power of the variable subset ω. As an exhaustive search of the whole variable space is not tractable when p is large (in microarray data $p > 5000$), stochastic approximations are proposed, see Gadat and Younes (2007); Lê Cao *et al.* (2007b) for the detailed theory of the model. At iteration i in the algorithm, the probability weight vector is updated with a gradient descent:

$$\mathbb{P}_{i+1} = \Pi_{\mathcal{S}}[\mathbb{P}_i - \alpha_i d_i]$$

where $\Pi_{\mathcal{S}}$ is the projection on the simplex of probability map on the set of variables, so that $\mathbb{P}_{i+1}$ remains a probability vector, α_i is the step of the gradient, and d_i is the stochastic approximation of the gradient (see below).

The whole process is repeated *iter.max* times and the final output is $\mathbb{P}_{iter.max}$, that indicates the importance of each variable in the data set. To obtain a variable selection, the user only needs to rank the variables according to their decreasing weights, and to choose the size of the selection.

2.2. General algorithm

Input: a data matrix of size $n \times p$ and the class values vector of size n.

Parameters: *iter.max*, total number of iterations and `mtry`, size of the variable subset ω.

Output: $\mathbb{P}_{iter.max}$, the weight vector of length p.

Initialize $\mathbb{P}_0 = [1/p, \ldots, 1/p]$ (uniform distribution on all variables)

For $i = 1$ to *iter.max*

1. Variables: draw a subset ω_i with respect to $\mathbb{P}_i$
2. Cases: draw a bootstrap sample B_i in $1, \ldots, n$ and define $\bar{B}_i$ the out-of-bag cases
3. Train the classifier on variables in ω_i and cases in B_i
4. Test the classifier on variables in ω_i and cases in $\bar{B}_i$, compute the classification error rate ϵ_i
5. Compute the drift vector d_i
6. Update $\mathbb{P}_{i+1} = \Pi_{\mathcal{S}}[\mathbb{P}_i - \alpha_i d_i]$

where for each iteration i:

- $d_i = \frac{C(\omega_i,.)\epsilon_i}{\mathbb{P}_i(.)}$ is the approximated gradient, and $C(\omega_i, k)$ is the number of occurrences of variable k in the subset ω_i, in case this variable is drawn more than one time in ω_i.
- $\Pi_{\mathcal{S}}$ is the projection on the simplex, so that $\sum_{j=1}^{p} \mathbb{P}_{ij} = 1$ and $\forall j \quad \mathbb{P}_{ij} \geq 0$, $j = 1, \ldots, p$.
- α_i is the step of the gradient descent, and is set to $\frac{1}{i+10}$.

2.3. Application

We applied OFW to two supervised algorithms: SVM and CART.

Support vector machines

SVM (Vapnik 1999) is a binary classifier that attempts to separate the cases by defining an optimal hyperplane between the 2 classes up to a consistency criterion. Linear kernel SVMs are performed here because of their good generalization ability compared to more complex kernels. For multiclass data, we applied OFW with the one-vs.-one SVM approach that is implemented in the **e1071** package (Meyer 2001; Dimitriadou *et al.* 2008). **ofw** hence depends on **e1071**. The user only needs to set the total number of iterations to perform (`nsvm`) and the size `mtry` of the subset ω to draw at each iteration (see Section 4.2 for tuning).

Classification and regression trees

OFW is applied with the multiclass classifier CART (Breiman *et al.* 1984) that is well adequate for multiclass problems. Following the example of Breiman (1996), the trees were aggregated (bagging) to overcome their unstable characteristic. Hence, several classification trees are constructed on different bootstrap samples and with different subsets ω. The approximated gradient is also slightly modified. The modified algorithm is as follows:

Input: data matrix of size $n \times p$ and the class values vector of size n.

Parameters: *iter.max*, total number of iterations, `mtry`, size of the variable subset ω and `ntree`, number of trees to aggregate.

Output: $\mathbb{P}_{iter.max}$ a weight vector of length p.

Initialize $\mathbb{P}_0 = [1/p, \ldots, 1/p]$ (uniform distribution on all variables)

For $i = 1$ to *iter.max*

1. **For** $b = 1$ to *ntree*
   (a) Variables: draw a subset ω_i^b with respect to $\mathbb{P}_i$
   (b) Cases: draw a bootstrap sample B_i^b in $1, \ldots, n$ and define $\bar{B}_i^b$ the out-of-bag cases
   (c) Train the classifier on variables in ω_i^b and cases in B_i^b
   (d) Test the classifier on variables in ω_i^b and cases in $\bar{B}_i^b$, compute the classification error rate ϵ_i^b
2. Compute the drift vector D_i
3. Update $\mathbb{P}_{i+1} = \Pi_{\mathcal{S}}[\mathbb{P}_i - \alpha_i D_i]$

where D_i is an averaged time version of the gradient d_i (see Lê Cao *et al.* 2007b).

Hence, as in random forests (Breiman 2001), `ntree` trees are constructed on `ntree` bootstrap samples. The only difference lies in the construction of the classification trees: instead of randomly selecting a variable subset to split each node of each tree (random forests), the variable subset is drawn with respect to the probability $\mathbb{P}_i$ to construct each tree.

In addition to choose the total number of iterations to perform (`nforest`) and the size `mtry` of the subset ω to draw at each iteration, the user needs to choose the number of aggregated trees `ntree` (see Section 4.2 for tuning).

2.4. Unbalanced multiclass

Challenge when data are unbalanced

Multiclass problems are often considered as an extension of 2-class problems. However this extension is not always straightforward, especially in microarray data context. Indeed, the data sets are often characterized by unbalanced classes with a small number of cases in at least one of the classes. This imbalance is often due to rare classes (e.g., a rare disease where patients are few) that are biologically of interest. Nevertheless, most algorithms do not perform well for such problems as they aim to minimize the overall error rate instead of focusing on the minority class.

Weighted procedure in OFW: wOFW

An efficient way to take into account the unbalanced characteristics of the data set is to weight the error rate ϵ_i according to the number samples of each class in the bootstrap sample. This allows for the penalization of a classification error made on the minority class and, therefore, put more weight on the variables that help classify this latter class instead of the majority one (Lê Cao *et al.* 2007a).

This weighted approach has been implemented in both versions of the algorithm, called ofwCART and ofwSVM, and also stands for the evaluation step (Step 2 in Figure 1 and see Section 4.4).

3. Implementation issues

ofw is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=ofw`. The R version $\geq$ 2.5.0 is needed to load the package **e1071** on which **ofw** depends.

ofw is a set of R and C functions to perform either ofwCART or ofwSVM and to evaluate the performances of both algorithms. Two classes of functions in R and C are implemented. Figure 1 provides a schematic view of the analysis of a data set with **ofw**. Each step in Figure 1 will be detailed in Section 4 on a small microarray data set.

The R environment is the only user interface. The R procedure calls a C subroutine, whose results are returned to R. There is no formula interface and the predictors can be specified as a matrix or a data frame via the `x` argument, with factor responses as a vector via the `y` argument. Note that **ofw** only performs classification and does not handle categorical

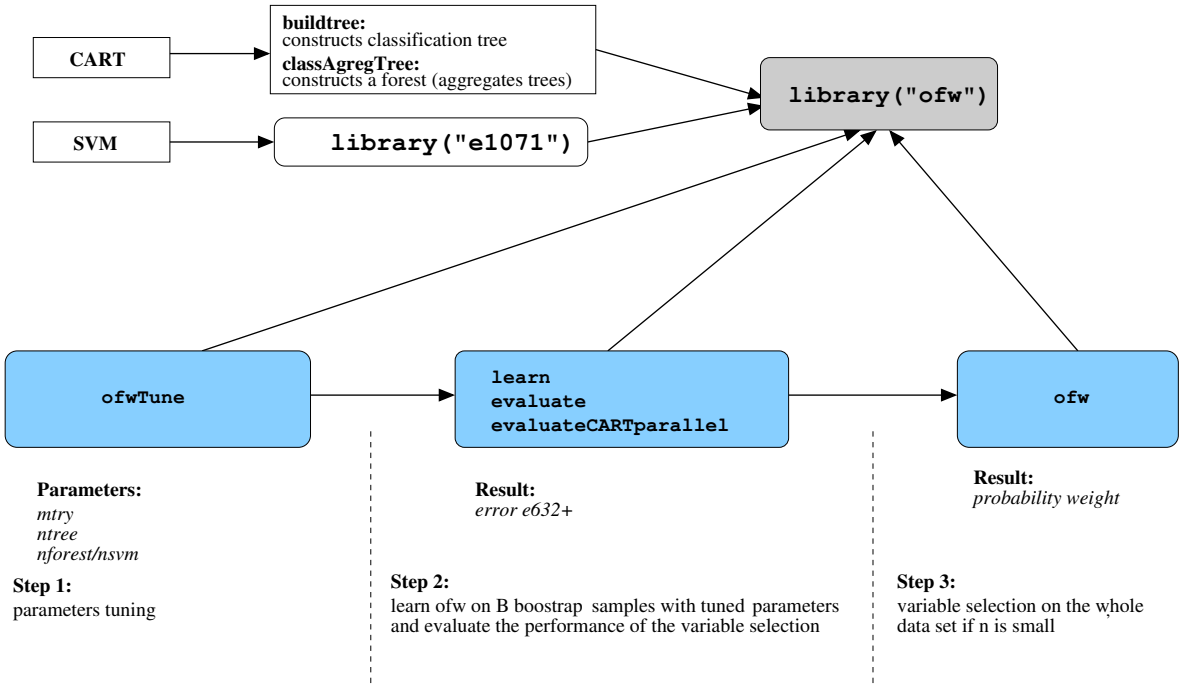

Figure 1: Schematic view of the data set analysis with **ofw**. The user only needs to use the R functions (in blue).

variables. Details of the components of each object from `ofwTune`, `ofw`, `learn`, `evaluate` and `evaluateCARTparallel` are provided in the online documentation. Methods provided for the classes `ofwTune` and `ofw` include `print`.

The C function `buildtree` that constructs classification trees has been borrowed from the Breiman and Cutler's Fortran programs and converted to C language. The function `classAgregTree` that aggregates trees was then largely inspired by the **randomForest** package (Liaw and Wiener 2002).

4. Using ofw

In the following, we detail the call to functions and R commands of **ofw**, that can be loaded into R by `library("ofw")`.

4.1. Illustrative data set

ofw was previously tested on several published microarray data sets (Lê Cao *et al.* 2007b,a) by comparing it with several other wrapper algorithms. We comment on the present paper the results obtained on one data set that is provided as an example in the package. SRBCT (Khan *et al.* 2001) is a data set of small round blue cell tumors of childhood. The training set consists of 63 training samples spanning 4 classes. The data set includes 2308 genes out of the 6567 after filtering for a minimal level of expression (performed by Khan *et al.* 2001). Further details about this data set can be found in `http://research.nhgri.nih.gov/microarray/`

Supplement. In order to minimize the computation time in this illustrative example, we have reduced SRBCT to 200 genes in the package by simply randomly selecting these out of the 2308 in the initial data set. We also added a factor `class` that indicates the class of each microarray sample. Note that normalization of the data, that is a crucial step in the analysis of microarray data is not dealt with **ofw** and has to be performed first by the user.

4.2. Tuning parameters

In the algorithm OFW, there are mainly 2–3 parameters to tune according to the applied classifier to ensure that OFW converges (Step 1 in Figure 1):

1. the size of the gene subset ω (called `mtry`).
2. the total number of iterations (called `nsvm` for ofwSVM and `nforest` for ofwCART).
3. the number of trees `ntree` to agregate for ofwCART.

The package **ofw** provides the function `ofwTune` to tune these parameters. Here is the command to launch `ofwTune` with ofwCART for different `mtry` values:

```
R> data("srbct")
R> attach(srbct)
R> tune.cart <- ofwTune(srbct, as.factor(class), type = "CART",
+    ntree = 150, nforest = 3000, mtry.test = seq(5, 25, length = 5),
+    do.trace = 100, nstable = 25)
R> detach(srbct)
```

Note that the only arbitrary parameter that is not tuned and has to be provided by the user is the number of variables `nstable` one wants to select (see below).

Tuning `mtry`

The function `ofwTune` consists in testing OFW (with CART or SVM) with several sizes of the subset ω (`mtry.test`). Then, for each `mtry.test`, OFW is performed twice, called `ofw1` and `ofw2`. The first `nstable` variables with the highest weights in $\mathbb{P}^{\texttt{ofw1}}_{\texttt{nforest}}$ and $\mathbb{P}^{\texttt{ofw2}}_{\texttt{nforest}}$ are extracted. The `ofwTune` function then outputs the intersection length of these two variable selections. For example, to tune the parameters with ofwCART, the command

```
R> tune.cart$param
        1  2  3  4  5
mtry    5 10 15 20 25
length 11  8  7  9  9
```

outputs the intersection length of the first `nstable` variables for each tested `mtry.test`. The value `mtry`= 5 gets the best stable result and should be chosen for Steps 2 and 3 in Figure 1 (evaluation and variable selection steps).

Early stopping

Instead of running OFW for all iterations in Step 1, the user can choose instead to set the number of variables (`nstable`) to select in the final variable selection step (Step 3). This halts

the algorithm once it becomes "stable", that is, when the `nstable` features of highest weights in $\mathbb{P}_i$ and $\mathbb{P}_{i+\texttt{do.trace}}$ are the same for iterations i and $i + \texttt{do.trace}$.

Finally, to choose the total number of iterations in Step 3, we simply suggest to take 2–3 times the number of iterations that were performed using the early stopping criterion, to ensure the convergence of the algorithm. This command outputs the number of iterations which were performed:

```
R> tune.cart$itermax
           1   2   3   4    5
ofwCART1 500 700 800 600 1000
ofwCART2 100 400 100 700 1100
```

Here the two algorithms ofwCART1 and ofwCART2 stopped at 500 and 100 iterations for `mtry` = 5. During the final learning step (Step 3), the user should hence set `nforest`$= 3 \cdot 500$.

Tuning `ntree` *(ofwCART)*

The best way to tune `ntree` would be then to run `ofwTune` with different values of `ntree` and choose the one that gets the largest intersection length of the first `nstable` variables. In our experience, the more numerous the trees, the more stable the results, usually for `ntree = 100` to `150`. The same stands for the weighted (`weight = TRUE`) or non-weighted (`weight = FALSE`) versions of OFW.

An example with ofwSVM

With the SVM classifier, the user has to specify `type = "SVM"` and use `nsvm` instead of `nforest` to indicate the number of chosen iterations. As SVMs are not aggregated, the user should set `nsvm` $\gg$ `nforest`.

```
R> tune.svm <- ofwTune(srbct, as.factor(class), type  ="SVM",
+    nsvm = 200000, mtry = 5, mtry.test = seq(5, 25, length = 5),
+    do.trace = 2000, nstable = 25)
R> tune.svm$param

       1  2  3  4  5
mtry   5 10 15 20 25
length 5  9  3  5  5

R> tune.svm$itermax

            1    2    3    4    5
ofwSVM1  6000 4000 8000 8000 4000
ofwSVM2 14000 8000 8000 8000 6000
```

In this case, with ofwSVM, the user should set `mtry = 10` and `nsvm = 24000` for the learning step if `nstable = 25`.

For both classifiers, we strongly advise to choose the smallest `mtry` that gives the more stable results. Our experience shows that for ofwCART, `mtry` will be rather small (5 to 15), as the

	#genes	#class.	#obs.	ofwCART	w-ofwCART	ofwSVM	w-ofwSVM
Lymphoma	4026	3	62	5^1	10^1	5	5
Leukemia	3000	3	72	5^1	5^1	15	10
SRBCT	2308	4	63	5^1	10^1	20	20
Brain	1963	5	42	5^1	25^1	10	10
Follicle	1564	3	42	10^2	10^2	25	25

Table 1: Values of the size of the subset ω. The number of trees aggregated is `ntree = 150` (1) and `ntree = 100` (2), respectively.

trees are aggregated. For ofwSVM, `mtry` can be larger (> 15). In both cases, `mtry` should not be greater than `nstable`, and, therefore, `mtryTest` $\leq$ `nstable`.

Table 1 illustrates the tuned parameters for several public data sets that were tested in Lê Cao *et al.* (2007a) for the weighted (ofwCART, ofwSVM) and non-weighted (w-ofwCART, w-ofwSVM) versions of OFW.

4.3. Variable selection and visualization plots

Once the parameters `mtry`, and `ntree` for ofwCART, have been chosen, the variable selection step (Step 3, Figure 1) can be performed, preferably on the whole data set if the sample size is too small, i.e., if n is roughly less than 80, or if the number of observations per class is too small. We advise to use the total number of iterations `nforest` or `nsvm`, rather than the `nstable` early stopping criterion to halt the algorithm, as suggested in Section 4.2.

The classifier to be applied has to be specified by the user. Here is the command for the variable selection step (Step 3) for ofwCART and ofwSVM.

```
R> learn.cart <- ofw(srbct, as.factor(class), type = "CART",
+    ntree = 150, nforest = 2500, mtry = 5)
R> learn.svm <- ofw(srbct, as.factor(class), type = "SVM",
+    nsvm = 30000, mtry = 5)
```

In the case of ofwCART, the evolution of the internal mean error rate $\bar{\epsilon}_i = \frac{1}{\text{ntree}} \sum_{b=1}^{\text{ntree}} \epsilon_i^b$ can be plotted for each iteration i, as shown in Figure 2, for $i = 1, \ldots,$ `nforest`:

```
R> plot(learn.cart$mean.error, type = "l")
```

The monotonic decreasing trend of $\bar{\epsilon}_i$ indicates if the parameters have been tuned correctly and thus if ofwCART converges. In the case of ofwSVM, the SVM are not aggregated, and the error variance is consequently very large: no decreasing trend can be observed and ϵ_i is not provided. Note that the internal error $\bar{\epsilon}_i$ does not evaluate the performance of OFW (see below Section 4.4) and is simply a way to assess the quality of the tuning.

One can also visualize the probability weights $\mathbb{P}_{\texttt{nforest}}$ or $\mathbb{P}_{\texttt{nsvm}}$ for each variable on Figures 3 (a) and (b):

```
R> plot(learn.cart$prob, type = "h")
R> plot(learn.svm$prob, type = "h")
```

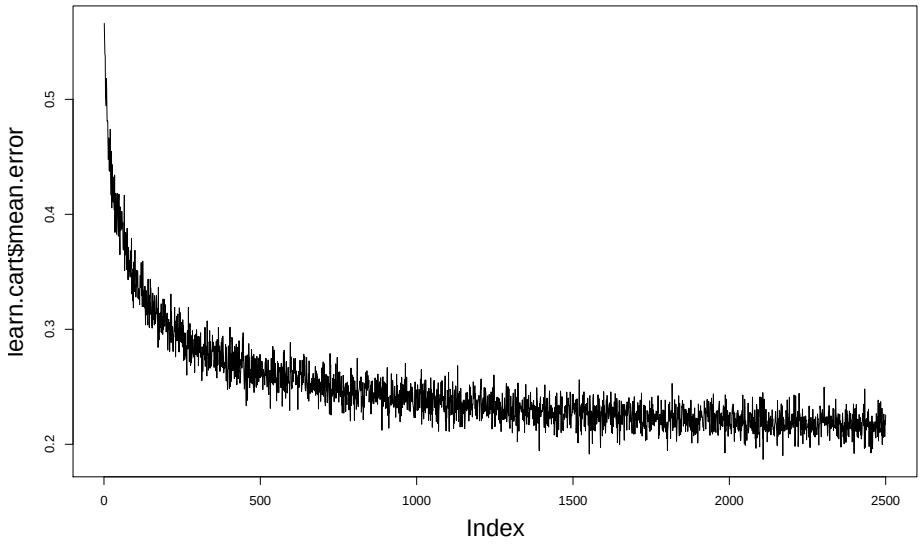

Figure 2: Internal mean error in ofwCART.

The selected variables can then be extracted by sorting the heaviest weights in $\mathbb{P}$, here for example for the 10 most discriminative variables with ofwCART:

```
R> names(learn.cart$list[1:10])
```

As $\mathbb{P}$ is a weight probability, the more numerous the variables, the smallest the weights on the variables. Hence, these weights are a qualitative rather than a quantitative importance measure of the variables, and the choice of a threshold is not advised. The different computations of the approximated gradient in ofwSVM (d_i) and in ofwCART (D_i), where $D_i \gg d_i$, actually lead to an important number of weights in $\mathbb{P}$ close to zero in ofwCART. Remark that some of the very discriminative variables get important weights in both methods. Usually, however, as the classifiers SVM and CART are differently constructed, the resulting variable selections will not be the same (see Lê Cao *et al.* 2007b,a).

4.4. Evaluation step

Method and implementation

To assess the performance of the variable selection performed by OFW (Step 2 in Figure 1), we propose to perform the e.632+ bootstrap error estimate from Efron and Tibshirani (1997) that is adequate for small sample size data sets (Ambroise and McLachlan 2002). Note that e.632+ does not dictate the optimal number of features to select. The error rate estimates that are computed with respect to the number of selected variables are only a way to compare the performances of different variable selection methods. Step 2 consists in two functions called `learn` and `evaluate`. The `learn` function simply learns OFW on a fixed number of bootstrap samples (`Bsample`) with the same tuned parameters defined in Step 1. The `evaluate` function

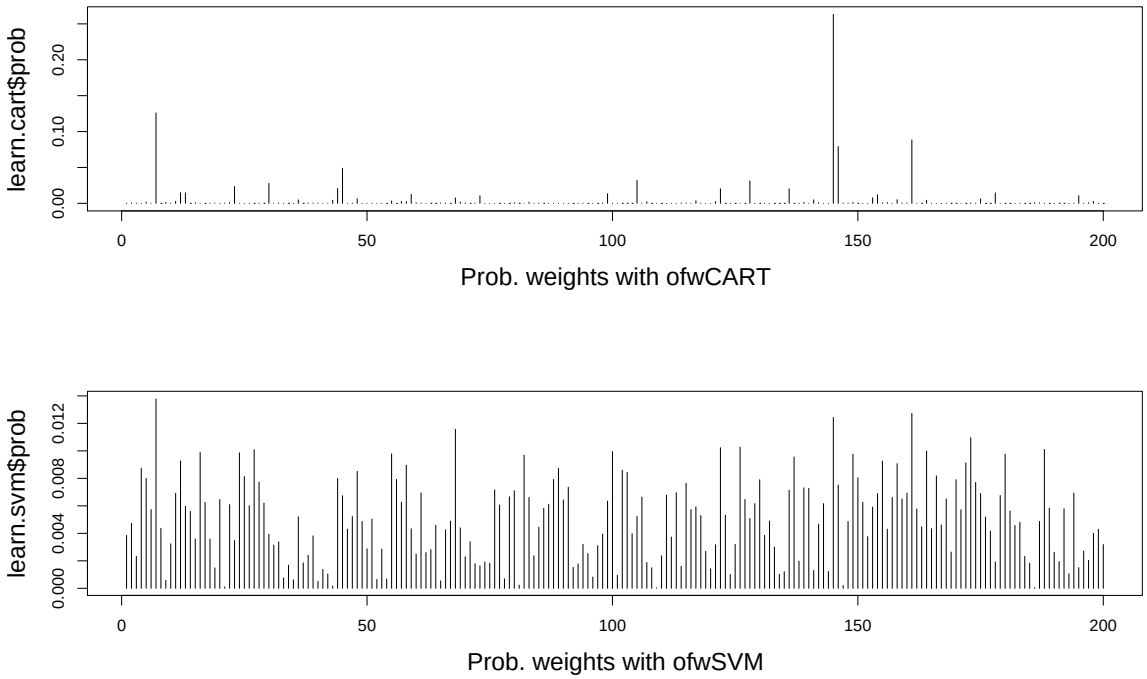

Figure 3: Variable weights that are computed with ofwCART (a) and ofwSVM (b).

that was inspired by the **ipred** package (Peters *et al.* 2002), computes and outputs the e.632+ error rate.

The `learn` *and* `evaluate` *functions*

```
R> learn.error.cart <- learn(srbct, as.factor(class), type = "CART",
+    ntree = 150, nforest = 2500, mtry = 5, Bsample = 10, do.trace = 100,
+    nstable = 25)
R> learn.error.svm <- learn(srbct, as.factor(class), type = "SVM",
+    nsvm = 30000, mtry = 5, Bsample = 10, do.trace = 2000, nstable = 25)
```

As the evaluation will be performed for a small selection size, we strongly advise to reduce the number of total iterations, using for example the early stopping criterion. In the literature, `Bsample` often equals to 10–50. On a 1.6 GHz 960 Mo RAM AMD Turion 64 X2 PC, the learning step of one bootstrap sample on a typical microarray data set ($p \simeq 5000$ and $n \simeq 50$) can take approximatively 2.5 hours. Hence, depending on the chosen value of `Bsample`, this evaluation step might be time consuming (see Section 5) and one can rather choose to perform parallel computing using the **Rmpi** package (Yu 2002, see supplementary file `v28i09-mpi.R`). If the SVM classifier is applied, each SVM is evaluated with the heaviest variables in $\mathbb{P}^b_{\texttt{nsvm}}$, which is learnt in the `learn` function, $b = 1, \ldots,$ `Bsample`. If the CART classifier is applied, the `evaluate` function aggregates `ntreeTest` trees. Each tree is constructed on a small

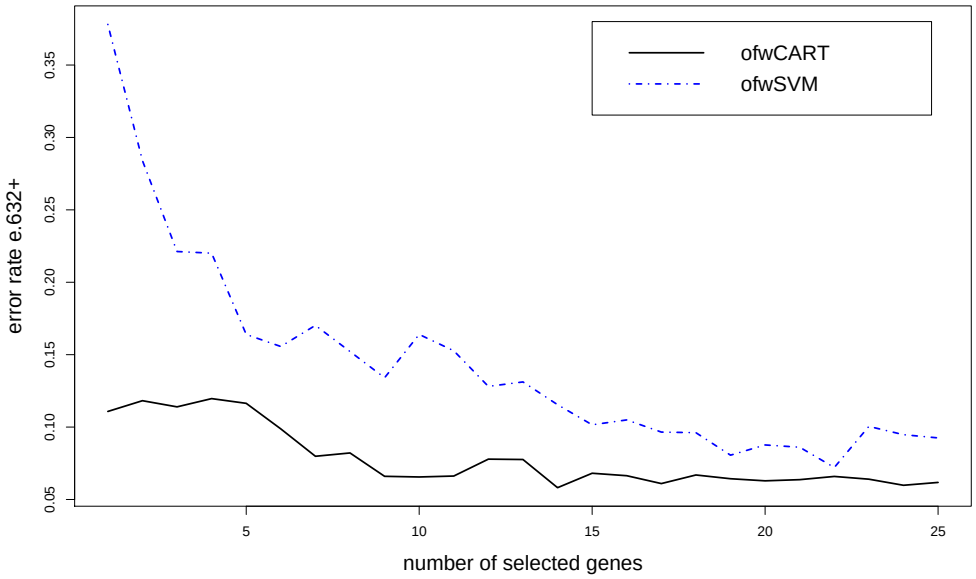

Figure 4: e.632+ error rate of ofwCART and ofwSVM.

variable subset that is randomly selected from the heaviest variables in $\mathbb{P}^b_{\texttt{nforest}}$, to avoid a too optimistic evaluation (see Lê Cao *et al.* 2007a). Both functions evaluate the variable selection of size `maxvar`:

```
R> eval.error.cart <- evaluate(learn.error.cart, ntreeTest = 100,
+    maxvar = 25)
R> eval.error.svm <- evaluate(learn.error.svm, maxvar = 25)
```

The `evalCARTparallel`function has also been implemented for parallel computing (refer to supplemental data). The aim of the `evaluate` function is to compare the performance of several algorithms (e.g., ofwCART and ofwSVM):

```
R> matplot(cbind(eval.error.cart$error, eval.error.svm$error),
+    type = "l", col = c(1, 4), lty = c(1, 4), lwd = 2, cex.lab = 1.3)
R> legend(18, 0.40, c("ofwCART", "ofwSVM"), col = c(1, 4),
+    lty = c(1, 4), cex = 1.2, lwd = 2)
```

Figure 4 displays the e.632+ bootstrap error rate of the selections resulting either from ofwCART or from ofwSVM with respect to the number of selected genes. In this example, where we compare the non-weighted versions of OFW, ofwCART seems to perform the best.

4.5. Further analysis: Comparing weighted and non-weighted OFW

The weighting procedure presented in Section 4.4 has also been included in the error evaluation function `evaluate`. To compare the two approaches, weighted (OFW) and non-weighted

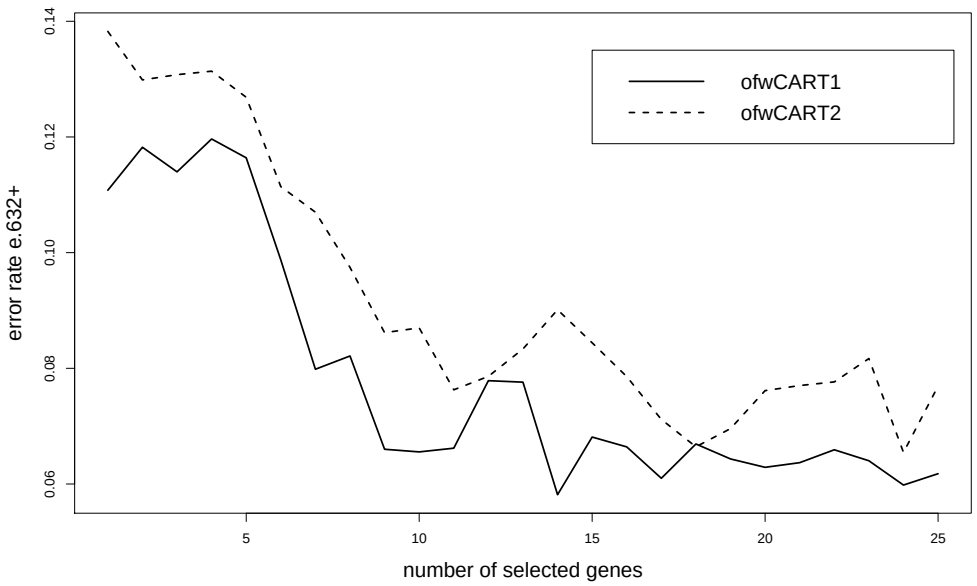

Figure 5: e.632+ error rate of ofwCART with e.632 not weighted (ofwCART1) and weighted (ofwCART2) .

(wOFW), we strongly advise to launch the `evaluate` function with the argument `weight = TRUE` in *both* cases to evaluate if the minority classes were misclassified or not. Otherwise, the e.632+ bootstrap error rate will always be lower for OFW than wOFW. This is illustrated in Figure 5 where the same gene selection resulting from ofwCART is evaluated either with the non-weighted version of e.632+ (ofwCART1) or with the weighted version of e.632+ (ofwCART2). Even though the same two gene selections are evaluated, the error rate is lower in ofwCART1 as this overall error rate only takes into account the microarrays that are rightly classified in the majoritary classes. In ofwCART2 where misclassified minoritary classes are taken into account, the error rate is consequently higher:

```
R> eval.error.cart1 = evaluate(learn.error.cart, ntreeTest = 100,
+    maxvar = 25)
R> eval.error.cart2 = evaluate(learn.error.cart, ntreeTest = 100,
+    maxvar = 25, weight = TRUE)
R> matplot(cbind(eval.error.cart1$error, eval.error.cart2$error),
+    type = "l", col = c(1, 1), lty = c(1, 2), lwd = 2, cex.lab = 1.3)
R> legend(18, 0.12, c("ofwCART1", "ofwCART2"), col = c(1, 1),
+    lty = c(1, 2), cex = 1.2, lwd = 2)
```

5. Computation time

Optimal feature weighting is a stochastic method that might be computationally time consuming if the variable dimension is very high. As the algorithm gets stabler for a large number

of iterations, the variable selection step (Step 3) might take 1–2 hours. Therefore, using parallel computing with the **Rmpi** package during the evaluation step (Step 2) might be advisable. If the dimension is considerable, we strongly advise to pre-filter the data set so as to remove uninformative variables that slow down the computation.

In this paper, on a very small microarray data set (200 genes), the tuning step (Step 1) took approximatively 20 minutes, the evaluation step (Step 2) 1.5 hours and the variable selection step (Step 3) 7 minutes.

6. Conclusion

We have implemented the stochastic algorithm OFW to select discriminative features. Although we illustrated this method on microarray data, OFW can be applied on any continuous data set for classification and prediction purposes.

Wrapper methods usually require heavy computation, and so does OFW. Efforts have thus been made to reduce some of the computation time by implementing C functions when applying CART and by proposing parallel programming during the learning step.

With this package, we hope to provide the user a method with a strong theoretical background that is easy to apply and that can bring interesting results in a feature selection framework.

Acknowledgments

We are grateful to "Projet Calcul en MIdi-Pyrenées" (CALMIP) for the intensive computations, and the anonymous reviewers for their helpful comments on the manuscript.

References

Ambroise C, McLachlan GJ (2002). "Selection Bias in Gene Extraction in Tumour Classification on Basis of Microarray Gene Expression Data." *Proceedings of the National Academy of Sciences*, **99**(1), 6562–6566.

Breiman L (1996). "Bagging Predictors." *Machine Learning*, **24**(2), 123–140.

Breiman L (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32.

Breiman L, Friedman JH, Olshen R, Stone C (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

Chen C, Liaw A, Breiman L (2004). "Using Random Forest to Learn Imbalanced Data." *Technical Report 666*, Department of Statistics, University of Berkeley. URL http://www.stat.berkeley.edu/tech-reports/666.pdf.

Chen D, Hua D, Reifman J, Cheng X (2003). "Gene Selection for Multi-Class Prediction of Microarray Data." In "CSB '03: Proceedings of the IEEE Computer Society Conference on Bioinformatics," p. 492. IEEE Computer Society, Washington, DC.

Dimitriadou E, Hornik K, Leisch F, Meyer D, Weingessel A (2008). ***e1071**: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.5-18, URL http://CRAN.R-project.org/package=e1071.

Efron B, Tibshirani R (1997). "Improvements on Cross-Validation: The e.632+ Bootstrap Method." *Journal of the American Statistical Association*, **92**, 548–560.

Gadat S, Younes L (2007). "A Stochastic Algorithm for Feature Selection in Pattern Recognition." *Journal of Machine Learning Research*, **8**, 509–547. ISSN 1533-7928.

Guyon I, Weston J, Barnhill S, Vapnik V (2002). "Gene Selection for Cancer Classification Using Support Vector Machines." *Machine Learning*, **46**(1-3), 389–422.

Khan J, Wei JS, Ringnér M, Saal LH, Ladanyi M, Westermann F, Berthold F, Schwab M, Antonescu CR, Peterson C, Meltzer PS (2001). "Classification and Diagnostic Prediction of Cancers Using Gene Expression Profiling and Artificial Neural Networks." *Nature Medicine*, **7**(6), 673–679.

Lê Cao KA, Bonnet A, Gadat S (2007a). "Multiclass Classification and Gene Selection with a Stochastic Algorithm." *Technical report*, Institut de Mathématiques, UMR CNRS 5219, University of Toulouse. URL http://www.lsp.ups-tlse.fr/Recherche/Publications/2007/cao05.pdf.

Lê Cao KA, Gonçalves O, Besse P, Gadat S (2007b). "Selection of Biologically Relevant Genes with a Wrapper Stochastic Algorithm." *Statistical Applications in Genetics and Molecular Biology*, **6**(1), Article 29. URL http://www.bepress.com/sagmb/vol6/iss1/art29/.

Li T, Zhang C, Ogihara M (2004). "A Comparative Study of Feature Selection and Multiclass Classification Methods for Tissue Classification Based on Gene Expression." *Bioinformatics*, **20**(15), 2429–2437.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." *R News*, **2**(3), 18–22. URL http://CRAN.R-project.org/doc/Rnews/.

Meyer D (2001). "Support Vector Machines." *R News*, **1**(3), 23–26. URL http://CRAN.R-project.org/doc/Rnews/.

Peters A, Hothorn T, Lausen B (2002). "**ipred**: Improved Predictors." *R News*, **2**(2), 33–36. URL http://CRAN.R-project.org/doc/Rnews/.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Vapnik VN (1999). *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer-Verlag, New York.

Weston J, Elisseeff A, Schölkopf B, Tipping M (2003). "Use of the Zero Norm with Linear Models and Kernel Methods." *Journal of Machine Learning Research*, **3**, 1439–1461.

Yeung KY, Burmgarner RE (2003). "Multi-Class Classification of Microarray Data with Repeated Measurements: Application to Cancer." *Genome Biology*, **4**(R83).

Yu H (2002). "**Rmpi**: Parallel Statistical Computing in R." *R News*, **2**(2), 10–14. URL http://CRAN.R-project.org/doc/Rnews/.

Affiliation:

Kim-Anh Lê Cao
Station d'Amélioration Génétique des Animaux (UR 631)
Institut National de la Recherche Agronomique
F-31326 Castanet, France
and
Institut de Mathématiques de Toulouse
UMR CNRS 5219
Université de Toulouse (UT3, CNRS INSA, UT1, UT2)
F-31062 Toulouse, France
E-mail: k.lecao@imb.uq.edu.au

Patrick Chabrier
Biométrie et Intelligence Artificielle (UR875)
Institut National de la Recherche Agronomique
F-31326 Castanet, France
E-mail: Patrick.Chabrier@toulouse.inra.fr

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 28, Issue 9 *Submitted:* 2008-02-13
November 2008 *Accepted:* 2008-07-25