Chapter 5
Variable Selection

Abstract This chapter is dedicated to variable selection using random forests: an automatic three-step procedure involving first a fairly coarse elimination of a large number of useless variables, followed by a finer and ascending sequential introduction of variables into random forest models, for interpretation and then for prediction. The principle and the procedure implemented in the VSURF package are presented on the spam dataset. The choice of VSURF parameters suitable for selection is then studied. In the final section, the variable selection procedure is applied to two real examples: predicting ozone concentration and analyzing genomic data.

5.1 Generalities

In the past, classic statistical problems typically involved many observations ($n =$, e.g., a few hundred or a few thousand) and relatively a few variables ($p =$ one to a few tens). Today, the ease of data acquisition has led to huge databases that collect new information almost daily. Traditional statistical techniques are poorly suited to processing these new quantities of data, in which the number of variables p can reach tens or even hundreds of thousands. At the same time, for many applications, the number of observations n can be reduced to a few tens, e.g., in the case of biomedical data. In this context, it is indeed common to gather many types of data on a given individual (e.g., gene expression data), but to keep the number of individuals on whom the experiment is conducted small (for the study of a disease, the number of affected individuals included in the study is often very limited). These data are said to be of high dimension: the number of variables is quite large in comparison to the number of observations, which is classically denoted by $n \ll p$. Here, we are referring to problems where n is several hundreds and p is several thousands. One of the most attractive features of random forests is that they are highly efficient both for traditional problems (where $p \leq n$) and for such high-dimensional problems. Indeed, RF have been previously shown to be inherently adapted to the high-dimensional case. For instance, Biau (2012) shows that if the true model meets certain sparsity conditions, then the RF predictor depends only on the active variables.

R. Genuer and J.-M. Poggi, *Random Forests with R*, Use R!,
https://doi.org/10.1007/978-3-030-56485-8_5

In many situations, in addition to designing a good predictor, practitioners also want additional information on the variables used in the problem. Statisticians are invited to propose a selection of variables in order to identify those that are most useful in explaining the input–output relationship. In this context, it is natural to think that relatively a few variables (say at most n and hopefully much less, for example, $\sqrt{n}$) actually affect the output, and it is necessary to make additional assumptions (called parsimony or sparsity) to make it tractable and meaningful. In Giraud (2014), there is a very complete presentation of mathematical problems and techniques for addressing this kind of questions.

Let us mention some methods for variable selection in high-dimensional contexts. Starting with an empirical study in Poggi and Tuleau (2006) where a method based on the variable importance index provided by the CART algorithm is introduced. In the same flavor, let us also mention Questier et al. (2005). Considering the problem more generally, Guyon et al. (2002), Rakotomamonjy (2003), and Ghattas and Ben Ishak (2008) use the score provided by the Support Vector Machines (SVM: Vapnik 2013) and Díaz-Uriarte and Alvarez De Andres (2006) propose a variable selection procedure based on the variable importance index related to random forests. These methods calculate a score for each of the variables, then perform a sequential introduction of variables (*forward* methods), or a sequential elimination of variables (*backward* or RFE for Recursive Feature Elimination methods), or perform step-by-step methods (*stepwise* methods) combining introduction and elimination of variables. In Fan and Lv (2008), a two-step method is proposed: a first step of eliminating variables to reach a reasonable situation where p is of the same order of magnitude of n, then a second step of model building using a forward strategy based, for example, on the Least Absolute Shrinkage and Selection Operator (Lasso: Tibshirani 1996). In this spirit, a general scheme for calculating an importance score for variables is proposed in Lê Cao et al. (2007), then the authors use this scheme with CART and SVM as the base method. Their idea is to learn a weight vector on all variables (their meta-algorithm is called Optimal Feature Weighting, OFW): a variable with a large weight is important, while a variable with a small weight is useless.

Finally, more recently, methods to improve Lasso for variable selection have been developed. The latter have points in common with the ensemble methods. Indeed, instead of trying to make selection "at once" with a classic Lasso, the idea is to construct several subsets of variables and then combine them. In Bolasso (for Bootstrap-enhanced Lasso), introduced by Bach (2008), several bootstrap samples are generated and then the Lasso method is applied to each of them. Bolasso is therefore to be compared with the Bagging of Breiman (1996). In Randomized Lasso, Meinshausen and Bühlmann (2010) propose to generate several samples by subsampling and add an additional random perturbation to the construction of the Lasso itself. Randomized Lasso is therefore to be compared to Random Forests-RI variant of random forests. In the same spirit, we can also mention Fellinghauer et al. (2013) which use RF for robust estimation in graphical models.

Interest in the subject still continues: for example, Hapfelmeier and Ulm (2012) propose a new selection approach using RF, and Cadenas et al. (2013) describe and compare these different approaches in a survey paper.

5.2 Principle

In Genuer et al. (2010b), we propose a variable selection method (see also in Genuer et al. 2015, the corresponding **VSURF** package). This is an automatic procedure in the sense that there is no a priori to make the selection. For example, it is not necessary to specify the desired number of variables; the procedure adapts to the data to provide the final subset of variables. The method involves two steps: the first, fairly coarse and descending, proceeds by thresholding the importance of the variables to eliminate a large number of useless variables, while the second, finer and ascending, consists of a sequential introduction of variables into random forest models.

In addition, we distinguish two variable selection objectives: interpretation and prediction (although this terminology may lead to confusion):

- For interpretation, we try to select all the variables X^j strongly related to the response variable Y (even if the variables X^j are correlated with each other).
- While for a prediction purpose, we try to select a parsimonious subset of variables sufficient to properly predict the response variable.

Typically, a subset built to satisfy the first objective may contain many variables, which will potentially be highly correlated with each other. On the contrary, a subset of variables satisfying the second one will contain a few variables, weakly correlated.

A situation illustrates the distinction between the two types of variable selection. Consider a high-dimensional classification problem ($n \ll p$) for which each explanatory variable is associated with a pixel in an image or a voxel in a 3D image as in brain activity classification (fMRI) problems; see, for example Genuer et al. (2010a). In such situations, it is natural to assume that many variables are useless or uninformative and that there are unknown groups of highly correlated predictors corresponding to regions of the brain involved in the response to a given stimulation. Although both variable selection objectives may be of interest in this case, it is clear that finding all the important variables highly related to the response variable is useful for interpretation, since the selected variables correspond to entire regions of the brain or of an image. Of course, the search for a small number of variables, sufficient for a good prediction, makes it possible to obtain the most discriminating variables in the regions previously highlighted but is of less priority in this context.

5.3 Procedure

In this section, we present the skeleton of the procedure before providing additional details, in the next section, after the application of the method to the `spam` data.

The first step is common to both objectives while the second depends on the goal:

- Step 1. Ranking and preliminary elimination:
  - Rank the variables by decreasing importance (in fact by average VI over typically 50 forests).
  - Eliminate the variables of low importance (let us denote m to be the number of retained variables).
    More precisely, starting from this order, we consider the corresponding sequence of standard deviations of the VIs that we use to estimate a threshold value on the VIs. Since the variability of the VIs is greater for the variables truly in the model than for the uninformative variables, the threshold value is given by estimating the standard deviation of the VI for the latter variables. This threshold is set at the minimum predicted value given by the CART model fitting the data (X, Y) where the Y are the standard deviations of the VI and the X are their ranks.
    Then only variables whose average importance VI is greater than this threshold are kept.
- Step 2. Variable selection:
  - For *interpretation*: we build the collection of nested models given by forests built on the data restricted to the first k variables (that is the k most important), for $k = 1$ to m, and we select the variables of the model leading to the lowest OOB error. Let us denote by m' the number of selected variables.
    More precisely, we calculate the averages (typically over 25 forests) of the OOB errors of the nested models starting with the one with only the most important variable and ending with the one involving all the important variables previously selected. Ideally, the variables of the model leading to the lowest OOB error are selected. In fact, to deal with instability, we use a classical trick: we select the smallest model with an error less than the lowest OOB error plus an estimate of the standard deviation of this error (based on the same 25 RF).
  - For *prediction*: from the variables selected for interpretation, a sequence of models is constructed by sequentially introducing the variables in increasing order of importance and iteratively testing them. The variables of the last model are finally selected.
    More precisely, the sequential introduction of variables is based on the following test: a variable is added only if the OOB error decreases more than a threshold. The idea is that the OOB error must decrease more than the average variation generated by the inclusion of non-informative variables. The threshold is set to the average of the absolute values of the first-order differences of the OOB errors between the models including m' variables and the one with m variables:

$$\frac{1}{m - m'} \sum_{j=m'}^{m-1} | \, errOOB(j+1) - errOOB(j) \, | \tag{5.1}$$

    where $errOOB(j)$ is the OOB error of the forest built with the j most important variables.

It should be stressed that all thresholds and reference values are calculated using only the data and do not have to be set in advance.

5.4 The VSURF Package

Let us start by illustrating the use of the **VSURF** package (Variable Selection Using Random Forests) on the simulated data *toys* introduced in Sect. 4.2 with $n = 100$ and $p = 200$, i.e., 6 true variables and 194 non-informative variables. The loading of the **VSURF** package as well as the `toys` data, included in the package, is done using the following commands:

```
> library(VSURF)
> data("toys")
```

The `VSURF()` function is the main function of the package and performs all the steps of the procedure. The random seed is fixed in order to obtain exactly the same results when applying later the procedure step by step:

```
> set.seed(3101318)
> vsurfToys <- VSURF(toys$x, toys$y, mtry = 100)
```

The methods `print()`, `summary()`, and `plot()` provide information on the results:

```
> summary(vsurfToys)

   VSURF computation time: 1.2 mins

   VSURF selected:
      34 variables at thresholding step (in 45 secs)
      4 variables at interpretation step (in 24.7 secs)
      3 variables at prediction step (in 1.2 secs)
```

```
> plot(vsurfToys)
```

Now, let us detail the main steps of the procedure using the results obtained on simulated `toys` data. Unless explicitly stated otherwise, all graphs refer to Fig. 5.1.

- Step 1.
  - Variable ranking.
    The result of the ranking of the variables is drawn on the graph at the top left. Informative variables are significantly more important than noise variables.
  - Variable elimination.
    From this ranking, we construct the curve of the corresponding standard deviations of VIs. This curve is used to estimate a threshold value for VIs. This

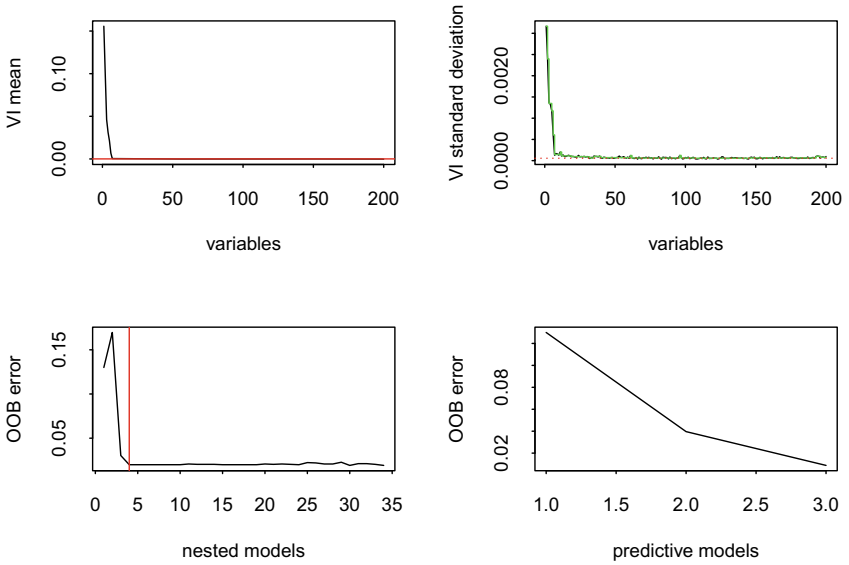

Fig. 5.1 Illustration of the results of the `VSURF()` function applied to the `toys` data

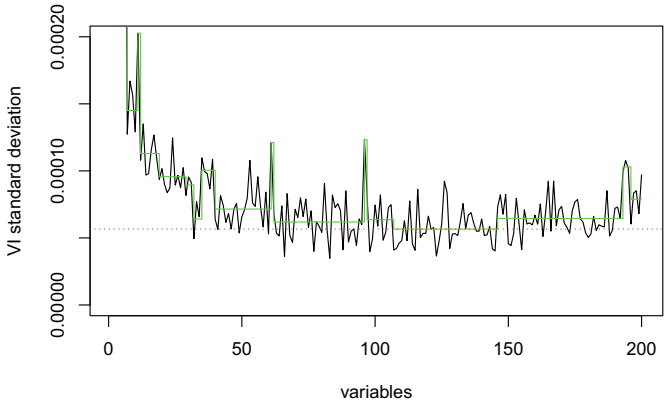

Fig. 5.2 Zoom of the top-right graph of Fig. 5.1

threshold (represented by the horizontal dotted red line in Fig. 5.2, which is a zoom of the top-right graph of Fig. 5.1), is set to the minimum predicted value given by a CART model fitted to this curve (see the piecewise constant green function on the same graph).

We then retain only the variables whose average VI exceeds this threshold, i.e., those whose VI is above the horizontal red line in the graph at the top left of Fig. 5.1.

The construction of forests and the ranking and elimination steps are obtained using the `VSURF_thres()` function:

```
> set.seed(3101318)
> vsurfThresToys <- VSURF_thres(toys$x, toys$y, mtry = 100)
```

The output of the `VSURF_thres()` function is a list containing all the results of this step. The main output arguments are `varselect.thres` which contains the indices of the variables selected at this step, `imp.mean.dec` and `imp.sd.dec` which contain the mean VI and the associated standard deviation (the order induced by the decreasing values of the mean VI is available in `imp.mean.dec.ind`).

```
> vsurfThresToys$varselect.thres
```

```
 [1]   3   2   6   5   1   4 184  37 138  81 159  17 180
[14] 191 131  94  52 165  96 192 157 198  21 111  25  29
[27]  12 109  64 107  70 186  46 188
```

Finally, Fig. 5.2 can be obtained directly from the object `vsurfToys` with the following command:

```
> plot(vsurfToys, step = "thres", imp.mean = FALSE,
    ylim = c(0, 2e-04))
```

We can see on the VI standard deviation curve (top-right graph of Fig. 5.1) that the standard deviation of the informative variables is large compared to that of the noise variables, which is close to zero.

- Step 2.
  - Procedure for selecting variables for interpretation.
    We calculate the OOB errors of random forests (on average over 25 repetitions) of nested models from the one with only the most important variable, and ending with the one with all the important variables stored previously.
    We select the smallest model with an OOB error less than the minimum OOB error increased by its empirical standard deviation (based on 25 repetitions).
    We use the `VSURF_interp()` function for this step. Note that we must specify the indices of the variables selected in the previous step, so we set the argument `vars` to `vsurfThresToys$varselect.thres`:

```
> vsurfInterpToys <- VSURF_interp(toys$x, toys$y,
    vars = vsurfThresToys$varselect.thres)
```

The list of results of the `VSURF_interp()` function gives access mainly to `varselect.interp` giving the variables selected by this step and `err.interp` containing the OOB errors of the nested RF models.

```
> vsurfInterpToys$varselect.interp
```

```
[1] 3 2 6 5
```

In the bottom-left graph, we see that the error is decreasing rapidly. It reaches almost its minimum when the first four true variables are included in the model (see the red vertical line), then it remains almost constant. The selected model contains the variables `V3`, `V2`, `V6`, and `V5`, which are four of the six true variables, while the real minimum is reached for 35 variables.
Note that, to ensure the quality of OOB error estimates (see Genuer et al. 2008) along nested RF models, the `mtry` parameter of the `randomForest()` function is set to its default value if k (the number of variables involved in the current RF model) is not greater than n, otherwise it is set to $k/3$.

– Variable selection procedure for prediction.
We perform a sequential introduction of variables with a test: a variable is added only if the accuracy gain exceeds a certain threshold. This is set so that the error reduction is significantly greater than the average variation obtained by adding noise variables.
We use the `VSURF_pred()` function for this step. We must specify the error rates and variables selected in the interpretation step, respectively, in `err.interp` and `varselect.interp` arguments:

```
> vsurfPredToys <- VSURF_pred(toys$x, toys$y,
    err.interp = vsurfInterpToys$err.interp,
    varselect.interp = vsurfInterpToys$varselect.interp)
```

The main outputs of the `VSURF_pred()` function are the variables selected by this last step, `varselect.pred`, and the OOB error rates of the RF models, `err.pred`.

```
> vsurfPredToys$varselect.pred
```

```
[1] 3 6 5
```

For `toys` data, the final model for prediction purposes only includes variables `V3`, `V6`, and `V5` (see the graph at the bottom right). The threshold is set to the average of the absolute values of the differences of OOB error between the model with the $m' = 4$ variables and the model with $m = 36$ variables.

Finally, it should be noted that `VSURF_thres()` and `VSURF_interp()` can be executed in parallel using the same syntax as `VSURF()` (by specifying `parallel = TRUE`), while the `VSURF_pred()` function is not parallelizable.

Let us end this section by applying `VSURF()` to `spam` data.

Even if it is a dataset of moderate size, the strategy proposed here is quite time-consuming, so we will use `VSURF()` by taking advantage of parallel capabilities:

```
> set.seed(923321, kind = "L'Ecuyer-CMRG")
> vsurfSpam <- VSURF(type ~ ., spamApp, parallel = TRUE,
    ncores = 3, clusterType = "FORK")
```

The option `parallel = TRUE` allows to run the procedure in parallel, and the argument `clusterType` sets the type of "cluster" used: it can be left by default most of the time but the option `"FORK"` (specific to Linux and MacOS systems), coupled with the option `kind = "L'Ecuyer-CMRG"` of the `set.seed()` function , allows reproducibility of results.

```
> summary(vsurfSpam)
```

```
VSURF computation time: 42.1 mins

VSURF selected:
   55 variables at thresholding step (in 12.7 mins)
   24 variables at interpretation step (in 20.6 mins)
   19 variables at prediction step (in 8.7 mins)

VSURF ran in parallel on a FORK cluster and used 3 cores
```

The overall calculation time is 42 min and the interpretation phase is the longest (half that of the total duration) while the other phases share the other half. The procedure identifies three sets of variables of decreasing size: 55, 24, and 19 and the results are summarized in Fig. 5.3.

```
> plot(vsurfSpam)
```

Let us focus on the 24 variables retained in the interpretation set. They are not surprising, at least for the first ones, but they are still numerous.

```
> colnames(spamApp[vsurfSpam$varselect.interp])
```

```
 [1] "remove"          "hp"              "capitalLong"
 [4] "charExclamation" "capitalAve"      "charDollar"
 [7] "capitalTotal"    "free"            "george"
[10] "num000"          "edu"             "your"
[13] "hpl"             "money"           "you"
[16] "our"             "business"        "num1999"
[19] "meeting"         "re"              "font"
[22] "num650"          "internet"        "receive"
```

If we move on to the 19 variables selected for prediction, there are hardly any fewer, but the ones that are eliminated, `num000`, `hpl`, `money`, `internet` and `receive`, are either weakly interesting (the last ones) or highly correlated with those retained (the other ones).

```
> colnames(spamApp[vsurfSpam$varselect.pred])
```

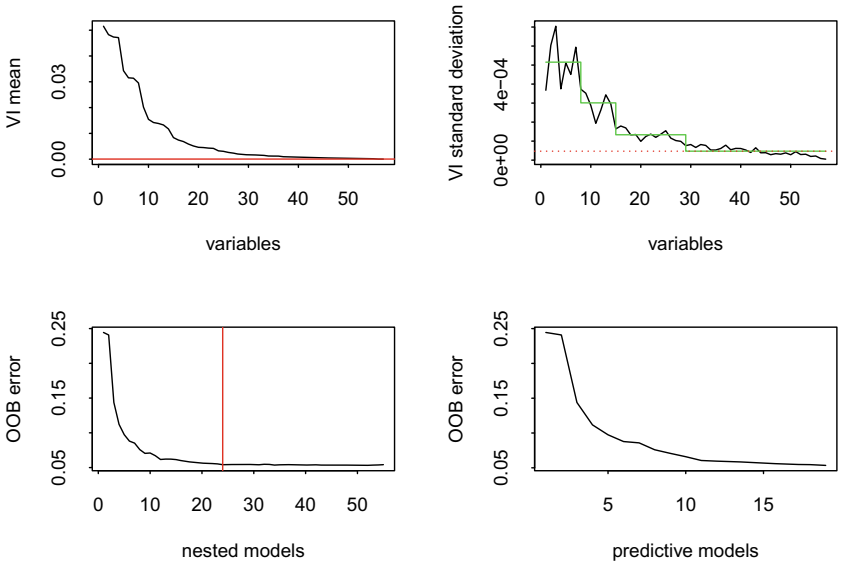

Fig. 5.3 Illustration of the results of `VSURF()`, `spam` data

```
 [1] "remove"          "hp"              "capitalLong"
 [4] "charExclamation" "capitalAve"      "charDollar"
 [7] "capitalTotal"    "free"            "george"
[10] "edu"             "your"            "you"
[13] "our"             "business"        "num1999"
[16] "meeting"         "re"              "font"
[19] "num650"
```

Nevertheless, it is clear that our procedure keeps too many variables and this is related to the too small value of the average jump for the example `spam`:

```
> vsurfSpam$mean.jump
[1] 0.0002715288
```

Multiplying this value by 15 (fixed after a trial and error process) gives a more satisfactory result with 8 variables sufficient for the prediction which are all significant except the last one `our`.

```
> set.seed(945834)
> vsurfSpamPred <- VSURF_pred(type ~ ., spamApp, nmj = 15,
    err.interp = vsurfSpam$err.interp,
    varselect.interp = vsurfSpam$varselect.interp)
```

```
> colnames(spamApp[vsurfSpamPred$varselect.pred])
```

```
[1] "remove"        "capitalLong"    "charExclamation" "capitalAve"
[5] "charDollar"    "free"           "george"          "our"
```

5.5 Parameter Setting for Selection

First of all, since `VSURF()` is strongly based on `randomForest()`, the two main parameters of this function (`mtry` and `ntree`) are taken over and have been kept the same name, so everything that applies to RF also applies to `VSURF()` for these parameters.

In addition, if you enter a value for another RF parameter, it is directly passed to the `randomForest()` function for all the RF built during the procedure. For example, if we add the option `maxnodes= 2` to the arguments of the `VSURF()` function, the whole procedure is performed with trees with 2 leaves.

```
> vsurfToysStump <- VSURF(toys$x, toys$y, mtry = 100, maxnodes = 2)
> summary(vsurfToysStump)
```

```
 VSURF computation time: 31.3 secs

 VSURF selected:
    14 variables at thresholding step (in 27.2 secs)
    8 variables at interpretation step (in 3 secs)
    2 variables at prediction step (in 1.1 secs)
```

```
> vsurfToysStump$varselect.interp
```

```
[1]   3   2   1   5   6 159  37 111
```

```
> vsurfToysStump$varselect.pred
```

```
[1] 3 5
```

There are also parameters specific to `VSURF()`:

- The number of trees in the forests for each of the three steps of the method: `nfor.thres` (which is the most important, because if it is taken too small, the estimated standard deviation at the thresholding step will be of bad quality; 50 by default), `nfor.interp`, and `nfor.pred` (25 by default, which stabilize the OOB error estimates for the last two steps).
- `nmin` (=**n**umber of **min**imum) sets the multiplying factor of the estimated standard deviation of the VI of a noise variable, to calculate the threshold value of the first step: "threshold = `min` $\times$ standard deviation of VI for noise variables". By default, it is set to 1, and increasing it amounts to a more restrictive thresholding and has the consequence of keeping fewer variables after the first step.
- `nsd` (=**n**umber of **s**tandard **d**eviation) allows to apply the rule "`nsd` SE rule" instead of applying the rule "1-SE rule" (introduced in Sect. 2.3). We would select fewer variables for the "interpretation" if we increase this value.

- `nmj` (=**n**umber of **m**ean **j**ump) is the multiplying factor of the mean jump due to the inclusion of a noise variable in the nested models in the last step.

Two functions allow to adjust the thresholding and interpretation steps without having to perform all the calculations again.

- First of all, a `tune()` method which, applied to the result of `VSURF_thres()`, allows to set the thresholding step. The parameter `nmin` (whose default value is 1) can be used to set the threshold to the minimum prediction value given by the CART model multiplied by `nmin`.

```
> vsurfThresToysTuned <- tune(vsurfThresToys, nmin = 3)
> vsurfThresToysTuned$varselect.thres
```

```
 [1]   3   2   6   5   1   4 184  37 138  81 159  17 180 191
```

We get 16 selected variables instead of 36 previously.

- Second, a `tune()` method which, applied to the result of `VSURF_interp()`, is of the same type and allows to set the interpretation step. If we now want to be more restrictive in our selection in the interpretation step, we can select the smallest model with an OOB error lower than the minimum OOB error plus an empirical standard deviation multiplied by `nsd` (with $\texttt{nsd} \geq 1$).

```
> vsurfInterpToysTuned <- tune(vsurfInterpToys, nsd = 5)
> vsurfInterpToysTuned$varselect.interp
```

```
[1] 3 2 6
```

We get 3 selected variables instead of 4 previously.

Finally, since the prediction step is a step-by-step process, to adjust this step, simply restart the `VSURF_pred()` function by changing the value of the parameter `nmj`.

```
> vsurfPredToysTuned <- VSURF_pred(toys$x, toys$y,
    err.interp = vsurfInterpToys$err.interp,
    varselect.interp = vsurfInterpToys$varselect.interp, nmj = 3)
> vsurfPredToysTuned$varselect.pred
```

```
[1] 3 6 5
```

5.6 Examples

5.6.1 Predicting Ozone Concentration

For a presentation of this dataset, see Sect. 1.5.2.

```
> library(VSURF)
> data("Ozone", package = "mlbench")
```

After loading the data, the result of the entire selection procedure is obtained by using the following command:

```
> set.seed(303601)
> OzVSURF <- VSURF(V4 ~ ., data = Ozone, na.action = na.omit)
> summary(OzVSURF)
```

```
VSURF computation time: 1.4 mins

VSURF selected:
   9 variables at thresholding step (in 50.7 secs)
   5 variables at interpretation step (in 21.6 secs)
   5 variables at prediction step (in 9.8 secs)
```

```
> plot(OzVSURF, var.names = TRUE)
```

Let us now examine these results successively (illustrated in Fig. 5.4). To reflect the order used in the definition of the variables, we first reorganize the variables at the end of the procedure.

```
> number <- c(1:3, 5:13)
> number[OzVSURF$varselect.thres]
```

```
[1]  9  8 12  1 11 10  5  7 13
```

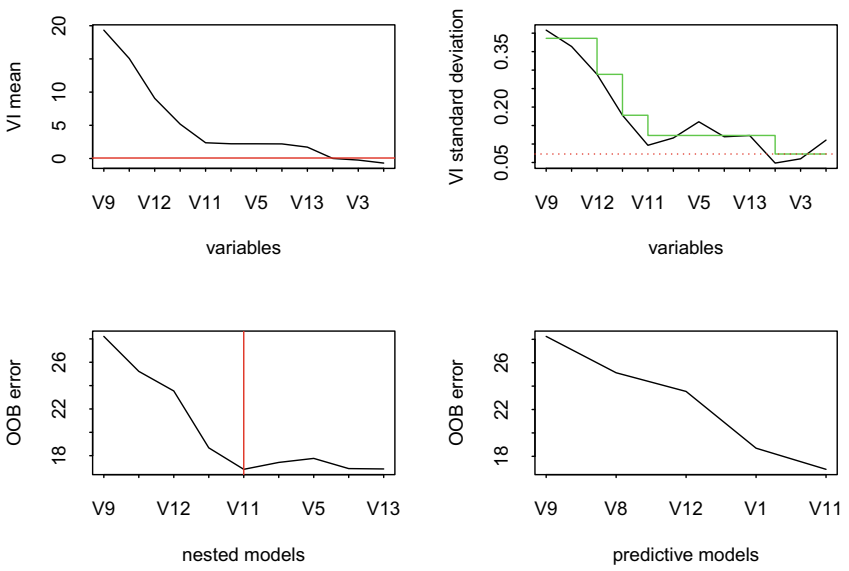

Fig. 5.4 Illustration of the results of `VSURF()`, `Ozone` data

After the first step, the 3 variables of negative importance (variables 6, 3, and 2) are eliminated as expected.

```
> number[OzVSURF$varselect.interp]
```

```
[1]  9  8 12  1 11
```

Then, the interpretation procedure leads to the selection of the 5-variable model, which contains all the most important variables.

```
> number[OzVSURF$varselect.pred]
```

```
[1]  9  8 12  1 11
```

With the default settings, the prediction step does not remove any additional variables.

In fact, our strategy more or less assumes that there exist some useless variables in the set of all initial variables, which is indeed the case in this dataset but not very significantly.

In addition, it should be noted here that our heuristics are clearly driven by prediction since the criterion for assessing the interest of a variable is closely related to the quality of the prediction or more exactly to its increasing after permutation.

5.6.2 *Analyzing Genomic Data*

For a presentation of this dataset, see Sect. 1.5.3.

Let us load the **VSURF** package, the `vac18` data, and then create an object `geneExpr` containing the gene expressions and an object `stimu` containing the stimuli to be predicted:

```
> library(VSURF)
> data("vac18", package = "mixOmics")
> geneExpr <- vac18$genes
> stimu <- vac18$stimulation
```

The global procedure with all parameters set to default values (note that the default value of `mtry` is $p/3$ even in classification, because as we have seen previously, the value of this parameter must be relatively high for high-dimensional problems) is obtained as follows:

```
> set.seed(481933)
> vacVSURF <- VSURF(x = geneExpr, y = stimu)
> summary(vacVSURF)
```

```
 VSURF computation time: 3.1 mins
```

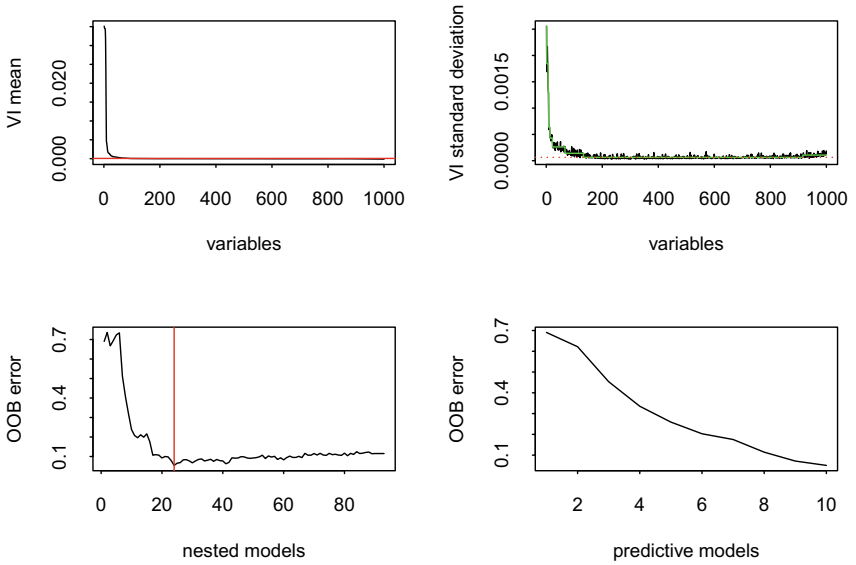

Fig. 5.5 Graphs illustrating the results of `VSURF()`, `Vac18` data

```
VSURF selected:
   93 variables at thresholding step (in 1.7 mins)
   24 variables at interpretation step (in 1.3 mins)
   10 variables at prediction step (in 7.9 secs)
```

```
> plot(vacVSURF)
```

The first thresholding step keeps only 93 variables. This is reasonable given the graph of the importance of the variables located at the top left of Fig. 5.5, which as pointed out in Sect. 4.5.3, illustrates a strong parsimony in the `Vac18` data.

The interpretation step of `VSURF()` leads to the selection of 24 variables, while the prediction step selects 10 variables.

Finally, the names of the variables (identifiers of the biochip probes used to measure gene expression) selected in the prediction step can be extracted as follows:

```
> probeSelPred <- colnames(geneExpr)[vacVSURF$varselect.pred]
> probeSelPred
```

```
 [1] "ILMN_1691156" "ILMN_2124802" "ILMN_2102693" "ILMN_1736939"
 [5] "ILMN_3252733" "ILMN_2188204" "ILMN_1658396" "ILMN_1663032"
 [9] "ILMN_3301824" "ILMN_2067444"
```

Computing time

The `VSURF()` function can be run in parallel using the following command:

```
> set.seed(627408, kind = "L'Ecuyer-CMRG")
> vacVSURFpara <- VSURF(x = geneExpr, y = stimu, parallel = TRUE,
    ncores = 3, clusterType = "FORK")
> summary(vacVSURFpara)
```

```
VSURF computation time: 1.5 mins

VSURF selected:
   97 variables at thresholding step (in 45.9 secs)
   23 variables at interpretation step (in 38 secs)
   13 variables at prediction step (in 7 secs)

VSURF ran in parallel on a PSOCK cluster and used 3 cores
```

We observe in this example a factor of about 2 in terms of saving execution time by using 3 cores instead of 1.