S3VS User Guide

Nilotpal Sanyal

January 09, 2026

Contents

1 Introduction and overview 1
1.1 What problem does S3VS solve? . . . 1
1.2 Model families supported . . . 2
1.3 S3VS in one picture . . . 2

2 Use of package functions with examples 2
2.1 Installation . . . 2
2.2 The main function S3VS() . . . 2
2.2.1 Example 1: Linear model . . . 4
2.2.2 Example 2: Binary classification model . . . 5
2.2.3 Example 3: Survival model . . . 6
2.3 Advanced usage: building blocks and customization . . . 7
2.3.1 Choosing leading variables: get_leadvars*() . . . 7
2.3.2 Constructing leading sets: get_leadsets() . . . 8
2.3.3 Selection within a set: VS_method*() . . . 8
2.3.4 Aggregating selections: select_vars() . . . 8
2.3.5 Removing variables: remove_vars() . . . 8
2.3.6 Response updating: update_y*() . . . 9
2.3.7 Stopping rule: looprun() . . . 9

3 Practical guidance and troubleshooting 9
3.1 Choosing method_sel and method_rem . . . 9
3.2 Highly correlated predictors: . . . 9
3.3 Computational tips for AFT: . . . 10
3.4 Reproducibility . . . 10

1 Introduction and overview

1.1 What problem does S3VS solve?

High-dimensional predictors with complex correlation structure are ubiquitous in modern applications (e.g., genomics, metabolomics, imaging, EHR features, text embeddings). In such settings, variable selection is challenging because:

- signals are sparse (few true effects among many candidates),
- predictors exhibit strong correlation structure (e.g., LD blocks, grouped features, multi-collinearity),
- model fitting can be computationally expensive (especially for survival models),
- different selection engines can yield unstable results if screening is naïve.

The **Structured Screen-and-Select Variable Selection (S3VS)** algorithm implemented in in the S3VS1 package addresses these issues by combining:

1. a structured screening step that proposes a small set of "leading" candidates, and
2. a within-set selection step (LASSO / SCAD / MCP / Bayesian nonlocal priors / survival-specific methods),
3. repeated in an iterative loop with principled aggregation rules.

This vignette introduces the main workflow, explains the key tuning knobs, and provides reproducible examples for each model family.

1.2 Model families supported

`S3VS` provides a unified workflow for:

- Linear models (`family = "normal"`)
- Binary generalized linear models (`family = "binomial"`)
- Survival models (`family = "survival"`) with:
  - Cox PH: `surv_model = "COX"`
  - AFT: `surv_model = "AFT"`

1.3 S3VS in one picture

At iteration t, let $\mathcal{V}_t$ denote the set of predictors not yet removed.

1. **Choose leading variables:** Compute a marginal association score s_j for each $X_j \in \mathcal{V}_t$ and using that score in a rule such as top-k or fixed thresholding, or percentage thresholding, keep a small set $\mathcal{L}_t \subset \mathcal{V}_t$.

2. **Build leading sets:** For each $\ell \in \mathcal{L}_t$, build a set $\mathcal{S}_\ell$ of predictors highly correlated with X_ℓ, using a rule such as top-k or fixed thresholding, or percentage thresholding.

3. **Select within each leading set:** Apply a variable selection engine to $\mathcal{S}_\ell$, producing selected/not-selected variables.

4. **Aggregate:** Aggregate selected and not-selected variables across leading sets using a conservative or liberal policy.

5. **Update & prune:** Optionally update the working response and remove non-selected variables.

6. **Stop:** terminate after m iterations or after `nskip` iterations with no selection.

2 Use of package functions with examples

2.1 Installation

If S3VS is available on CRAN, you can install it in the usual way:

```r
install.packages("S3VS")
```

If you have a source tarball (e.g., `S3VS_1.0.tar.gz`), install from source:

```r
install.packages("S3VS_1.0.tar.gz", repos = NULL, type = "source")
```

Then load the package:

```r
library(S3VS)
```

2.2 The main function `S3VS()`

`S3VS()` is the main and the top-level function of this package. Depending on the family, `S3VS()` dispatches to `S3VS_LM()`, `S3VS_GLM()`, or `S3VS_SURV()`, each of which calls the following functions to perform the whole analysis:

- `get_leadvars*()` for choosing leading variables,
- `get_leadsets()` for construting leading sets,
- `VS_method*()` for within-set variable selection,
- `select_vars()` for aggregating selected variables across sets,
- `remove_vars()` for aggregating not-selected variables across sets,
- `update_y*()` for updating the working response, and
- `looprun()` to check when the algorithm should be terminated.

The `S3VS()` function signature below shows the available options for each argument and default options.

```
S3VS(
  y, X,
  family = c("normal", "binomial", "survival"),
  cor_xy = NULL,
  surv_model = c("COX", "AFT"),
  method_xy = c("topk", "fixedthresh", "percthresh"),
  param_xy,
  method_xx = c("topk", "fixedthresh", "percthresh"),
  param_xx,
  vsel_method = NULL,
  method_sel = c("conservative", "liberal"),
  method_rem = c("conservative_begin", "conservative_end", "liberal"),
  sel_regout = FALSE,
  rem_regout = FALSE,
  update_y_thresh = 0.5,
  m = 100,
  nskip = 3,
  verbose = FALSE,
  seed = NULL,
  parallel = FALSE
)
```

- The rule for choosing the leading variables is specified by the arguments `method_xy` and `param_xy`, whereas the rule for determining the leading sets is specified by `method_xx` and `param_xx`. The marginal association measure for choosing the leading variables depends on family (correlation for “normal”, eta-squared for “binomial”, and marginal utility for “survival”), whereas Pearson correlation is used for determining the leading sets. The option “topk” keeps the predictors with the largest k association values, “fixedthresh” keeps predictors whose association is greater than or equal to a specified threshold, and “percthresh” keeps predictors whose association is within a given percentage of the best.

- The `vsel_method` argument specifies the within-set variable selection engine. The choices include “NLP”, “ENET”, “LASSO”, “SCAD”, and “MCP” for linear and generalized linear models; “LASSO” and “ENET” for survival “COX” models; and “AFTGEE”, “BRIDGE”, “PVAFT” for survival “AFT” models.

- The `method_sel` argument specifies the policy for aggregating predictors selected across leading sets in an iteration. The option “conservative” selects the smallest admissible set of predictors by intersecting the selected sets of predictors across leading sets, beginning with all and gradually reducing from the end until a non-empty intersection is found; this ensures only predictors consistently selected across leading sets are retained. The option “liberal” selects the largest admissible set of predictors by taking the union of all selected sets of predictors, so any predictor chosen in at least one leading set is included. If no predictor is selected from the first leading set, the iteration does not contribute to final selection and exclusion rules (method_rem) are applied instead.

- The `method_rem` argument specifies the policy for excluding predictors when no selections are made in an iteration. The “conservative_begin” option excludes the smallest admissible set of predictors by intersecting the non-selected sets of predictors starting from the first leading set; “conservative_end” does the same but begins from the last leading set and moves backward; "liberal" excludes the largest admissible set of predictors by taking the union of all non-selected sets of predictor. Predictors excluded under this rule are removed from subsequent iterations.

- `sel_regout` and `rem_regout` control whether the working response is updated during the S3VS iterations to help uncover weaker signals after accounting for stronger effects. When `sel_regout = TRUE` (GLM only), the response is updated using the predictors selected in the current iteration via `update_y_GLM()`, effectively conditioning on newly selected variables before the next screening step. When `rem_regout = TRUE` (LM and GLM), and an iteration results in no new selections but some predictors are removed, the response is updated using the removed predictors through `update_y_LM()` or `update_y_GLM()`, allowing the algorithm to adjust for their influence before continuing. Both options are ignored for model families where they do not apply.

Below, we provide examples of the usage of `S3VS()` for linear, generalized linear, and survival models.

2.2.1 Example 1: Linear model

We simulate a correlated design and a sparse linear signal.

```r
set.seed(1)

n <- 200
p <- 300

rho <- 0.6
Sigma <- rho ^ abs(outer(1:p, 1:p, "-"))
X_lm <- matrix(rnorm(n * p), n, p) %*% chol(Sigma)
colnames(X_lm) <- paste0("X", seq_len(p))

beta <- rep(0, p)
active <- c(5, 17, 50, 120, 201)
beta[active] <- c(2.0, -1.5, 1.2, 1.8, -2.2)

y_lm <- as.numeric(X_lm %*% beta + rnorm(n, sd = 2))
```

Next, we run un S3VS with LASSO within leading sets

```r
fit_lm <- S3VS(
  y = y_lm,
  X = X_lm,
  family = "normal",
  method_xy = "percthresh",
  param_xy = list(thresh = 95),
  method_xx = "topk",
  param_xx = list(k = 10),
  vsel_method = "LASSO",
  method_sel = "conservative",
  method_rem = "conservative_begin",
  verbose = FALSE,
  seed = 1
)
#> =================================
#> Number of selected variables: 6
#> Time taken: 0.3 sec
#> =================================
```

The selected variables are

```
fit_lm$selected
#> [1] "X5"   "X201" "X17"  "X120" "X50"  "X270"
```

Iteration-wise selected variables are

```
str(fit_lm$selected_iterwise)
#> List of 9
#>  $ : chr "X5"
#>  $ : chr "X201"
#>  $ : chr "X17"
#>  $ : chr "X120"
#>  $ : chr "X50"
#>  $ : chr "X270"
#>  $ : chr ""
#>  $ : chr ""
#>  $ : chr ""
```

`pred_S3VS()` refits a model on the selected variables using the method appropriate for the family.

```
Xsel <- X_lm[, fit_lm$selected, drop = FALSE]
pred_lm <- pred_S3VS(y = y_lm, X = Xsel, family = "normal", method = "LASSO")

head(pred_lm$y.pred)
#> [1]  2.4719666 -3.5834730 -3.9231277  2.7764653 -7.4049709  0.8373651
```

2.2.2 Example 2: Binary classification model

We simulate a correlated design and a sparse binary signal.

```
set.seed(2)

n <- 250
p <- 400

rho <- 0.4
Sigma <- rho ^ abs(outer(1:p, 1:p, "-"))
X_glm <- matrix(rnorm(n * p), n, p) %*% chol(Sigma)
colnames(X_glm) <- paste0("X", seq_len(p))

beta <- rep(0, p)
active <- c(10, 25, 90, 200)
beta[active] <- c(1.2, -1.0, 0.9, -1.4)

eta <- as.numeric(X_glm %*% beta)
pr  <- 1 / (1 + exp(-eta))
y_glm   <- rbinom(n, 1, pr)
```

We run un S3VS with SCAD within leading sets

```
fit_glm <- S3VS(
  y = y_glm,
  X = X_glm,
  family = "binomial",
  method_xy = "_survtopk",
  param_xy = list(k = 2),
  method_xx = "percthresh",
  param_xx = list(thresh = 90),
  vsel_method = "SCAD",
  sel_regout = TRUE,
  rem_regout = TRUE,
  update_y_thresh = 0.5,
  verbose = FALSE,
  seed = 1
)
#> =================================
#> Number of selected variables: 9
#> Time taken: 0.05 sec
#> =================================

fit_glm$selected
#> [1] "X200" "X25"  "X10"  "X277" "X101" "X195" "X90"  "X340" "X116"
```

2.2.3 Example 3: Survival model

For survival models, y must be a list with components:

- `time`: observed times
- `status`: event indicator (1 = event, 0 = censored)

We generate a survival data as follows.

```
set.seed(3)

n <- 300
p <- 200
X_surv <- matrix(rnorm(n*p), n, p)
colnames(X_surv) <- paste0("X", seq_len(p))

beta <- rep(0, p)
active <- c(3, 30, 77, 150)
beta[active] <- c(0.6, -0.5, 0.7, -0.8)

linpred <- as.numeric(X_surv %*% beta)
base_rate <- 0.05

time <- rexp(n, rate = base_rate * exp(linpred))
cens <- rexp(n, rate = 0.02)
status <- as.integer(time <= cens)
time <- pmin(time, cens)

y_surv <- list(time = time, status = status)
```

First, we run S3VS assuming a Cox model.

```
fit_cox <- S3VS(
  y = y_surv,
  X = X_surv,
  family = "survival",
  surv_model = "COX",
  method_xy = "topk",
  param_xy = list(k = 2),
  method_xx = "percthresh",
  param_xx = list(thresh = 90),
  vsel_method = "LASSO",
  verbose = FALSE,
  seed = 1
)
#> ==================================
#> Number of selected variables: 4
#> Time taken: 0.38 sec
#> ==================================

fit_cox$selected
#> [1] "X150" "X77"  "X3"   "X30"
```

Next, we run S3VS assuming an AFT model. AFT screening is often more expensive because adding a candidate can require repeated model fitting. Start with smaller `k` values.

```
fit_aft <- S3VS(
  y = y_surv,
  X = X_surv,
  family = "survival",
  surv_model = "AFT",
  method_xy = "topk",
  param_xy = list(k = 2),
  method_xx = "topk",
  param_xx = list(k = 2),
  vsel_method = "AFTGEE",
  verbose = FALSE,
  seed = 1,
  parallel = FALSE
)
#> ==================================
#> Number of selected variables: 8
#> Time taken: 7.01 sec
#> ==================================

fit_aft$selected
#> [1] "X150" "X77"  "X3"   "X30"  "X148" "X74"  "X62"  "X116"
```

2.3 Advanced usage: building blocks and customization

This section individually discusses the helper functions that the top-level `S3VS*()` functions call.

2.3.1 Choosing leading variables: `get_leadvars*()`

The generic `get_leadvars()` dispatches to family-specific implementations `get_leadvars_LM()`, `get_leadvars_GLM()` and `get_leadvars_SURV()`.

Typical usages are given below:

```
# For the earlier linear model example
leadvars <- get_leadvars(y = y_lm, X = X_lm, family = "normal", varsleft = colnames(X),
                        method = "topk", param = list(k = 20))
leadvars
#>  [1] "X5"   "X201" "X120" "X4"   "X202" "X17"  "X200" "X50"  "X121" "X49"
#> [11] "X119" "X51"  "X6"   "X203" "X36"  "X122" "X18"  "X3"   "X102" "X204"
```

```
# For the earlier generalized linear model example
leadvars <- get_leadvars_GLM(y = y_glm, X = X_glm, method = "percetasqthresh",
                             param = list(thresh = 80))
leadvars
#> [1] "X200" "X25"

# For the earlier survival Cox model
leadvars <- get_leadvars_SURV(y = y_surv, X = X_surv, method = "topk",
                              param = list(k = 5))
leadvars
#> [1] "X150" "X77"  "X30"  "X3"   "X120"
```

For linear models, if you need to reuse association scores across multiple runs (e.g., repeated stability selection), compute `cor_xy` once and pass it to `S3VS()`:

```
cor_xy <- abs(cor(y, X))
fit <- S3VS(y = y, X = X, family = "normal", cor_xy = cor_xy, ...)
```

2.3.2 Constructing leading sets: `get_leadsets()`

Once leading variables are chosen, `get_leadsets()` builds correlation neighborhoods.

```
leadsets <- get_leadsets(X = X, leadvars = leadvars,
                         method_xx = "topk", param_xx = list(k = 40))
```

2.3.3 Selection within a set: `VS_method*()`

The generic `VS_method()` dispatches to family-specific implementations `VS_method_LM()`, `VS_method_GLM()` and `VS_method_SURV()`. Typical examples follow.

```
sel <- VS_method_LM(y = y, X = X[, leadsets[[1]], drop = FALSE],
                    vsel_method = "SCAD")
sel$selected

sel <- VS_method_GLM(y = y, X = X[, leadsets[[1]], drop = FALSE],
                    vsel_method = "MCP")
sel$selected

sel <- VS_method_SURV(y = y, X = X[, leadsets[[1]], drop = FALSE], surv_model = "AFT",
                    vsel_method = "AFTGEE")
sel$selected
```

2.3.4 Aggregating selections: `select_vars()`

When different leading sets yield different selections, `select_vars()` reconciles them. Typical examples follow.

```
agg <- select_vars(sel_list, method_sel = "conservative")
agg
```

Whereas `method_sel = "conservative"` emphasizes stability/consistency,`method_sel = "liberal"`: emphasizes sensitivity (union-like behavior)

2.3.5 Removing variables: `remove_vars()`

If no variables are selected at an iteration, removal rules can be applied to shrink the search space.

```
rem <- remove_vars(varsleft, varsselected, method_rem = "conservative_begin")
rem
```

2.3.6 Response updating: `update_y*()`

To help detect weaker signals after strong effects are accounted for, S3VS can update the working response. Specifically, `update_y_LM()` provides regression residualization for LM and `update_y_GLM()` uses probability updates with a threshold (controlled by `update_y_thresh`)for GLM.

```
y_new <- update_y_LM(y = y, X = X[, varsselected, drop = FALSE])

y_new <- update_y_GLM(y = y, X = X[, varsselected, drop = FALSE], update_y_thresh = 0.4)
```

2.3.7 Stopping rule: `looprun()`

The function `looprun()` implements the stopping rule based on maximum iterations `m`, maximum allowed "no progress" iterations `nskip` unless `varsleft` is exhausted.

```
looprun(varsselected, varsleft, max_nocollect = 2, m = 100, nskip = 3)
```

3 Practical guidance and troubleshooting

Some practical guidelines for specific usages of the S3VS are given below.

3.1 Choosing `method_sel` and `method_rem`

The choice of `method_sel` and `method_rem` governs how selections are aggregated across leading sets and how aggressively variables are removed from the search space, and therefore reflects a fundamental trade-off between stability and sensitivity. If the primary goal is to control false discoveries and obtain a parsimonious, interpretable model, it is advisable to begin with `method_sel = "conservative"` and `method_rem = "conservative_begin"`, which emphasize consistency across leading sets and remove variables cautiously, particularly in early iterations. This configuration tends to favor predictors that are repeatedly supported by the data and yields more stable selections. In contrast, if the goal is to maximize sensitivity and identify a broader pool of potentially relevant candidates—such as in exploratory analyses or hypothesis generation—one may opt for `method_sel = "liberal"` and `method_rem = "liberal"`. These choices relax aggregation and removal criteria, allowing more variables to be retained and carried forward, at the cost of potentially increased false positives.

3.2 Highly correlated predictors:

When predictors are extremely correlated, careful tuning of the leading-set construction and the choice of selection engine becomes especially important. In such settings, it is often beneficial to increase `param_xx`, which enlarges the leading sets and allows groups of highly correlated variables to be examined jointly rather than in isolation. This can improve stability and reduce the risk of arbitrarily selecting one variable from a strongly correlated cluster while excluding equally relevant alternatives. At the same time, the size of leading sets should be chosen with computational feasibility in mind, since larger sets increase the cost of within-set selection. For linear models, penalization methods such as `"SCAD"` or `"MCP"` are worth considering, as they tend to exhibit different shrinkage and grouping behavior than `"LASSO"` and may yield more stable selections in the presence of strong collinearity. If a Bayesian approach is preferred, the `"NLP"` option provides a selection engine based on nonlocal priors, which enforce stronger sparsity and can be particularly effective at distinguishing true signals from correlated noise.

3.3 Computational tips for AFT:

For AFT models, the screening and selection steps in S3VS can be computationally demanding because evaluating candidate predictors often requires repeated model fitting and likelihood comparisons. To keep computations tractable, it is advisable to start with small screening and leading-set sizes (e.g., modest `k` values in both `param_xy` and `param_xx`) and increase them only if important signals appear to be missed. Whenever possible, enable parallel computation (`parallel = TRUE` when you have `future.apply` configured) to distribute candidate evaluations across multiple cores, particularly during the AFT screening step, which is typically the dominant cost. Using simpler or faster AFT selection engines (such as `AFTGEE`) in early exploratory runs can help identify promising regions of the predictor space before switching to more complex methods. Careful monitoring of iteration-wise progress and early stopping when no new variables are being selected can further reduce unnecessary computation, making the overall AFT workflow more efficient and scalable.

```r
library(future)
plan(multisession)
fit_aft <- S3VS(..., family="survival", surv_model="AFT", parallel=TRUE)
```

3.4 Reproducibility

For reproducibility, users should set the seed argument (for example, `seed = 1`) to ensure that results are repeatable across runs. It is also recommended to save the full object returned by S3VS(), as it contains all essential information needed for transparency and auditability, including the final set of selected variables (`selected`), the iteration-wise selection history (`selected_iterwise`), and the total runtime (`runtime`).