Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

SPAR: Sparse Projected Averaged Regression in R

Roman Parzer
TU Wien

Laura Vana Gür
TU Wien

Peter Filzmoser
TU Wien

Abstract

SPAR is a newly developed package for R that builds ensembles of predictive generalized linear models (GLMs) with high-dimensional (HD) predictors. It employs an algorithm utilizing variable screening and random projection tools to efficiently handle the computational challenges associated with large sets of predictors. The package is designed with a strong focus on extensibility. Screening and random projection techniques are implemented as S3 classes with user-friendly constructor functions, enabling users to easily integrate and develop new procedures. This design enhances the package's adaptability and makes it a powerful tool for a variety of high-dimensional applications.

Keywords: random projection, variable screening, ensemble learning, R.

1. Introduction

SPAR is a new package for R (R Core Team 2024) designed to build ensembles of predictive generalized linear models (GLMs) for high-dimensional settings where $p > n$ or even $p >> n$ by integrating variable screening and random projection techniques to achieve dimensionality reduction of the predictors. The implemented algorithm provides a computationally efficient procedure for high-dimensional regression, accommodating for both discrete and continuous responses. The package is designed with flexibility in mind, providing users with a versatile framework to extend the implemented screening and projection techniques with their own custom procedures. This is facilitated by leveraging R's S3 classes, making the process convenient and user-friendly. The adaptability of the algorithm and the extensibility of the package to combine different techniques enhance **SPAR**'s suitability across a range of high-dimensional applications.

Random projection is a computationally-efficient method which linearly maps a set of points in high dimensions into a much lower-dimensional space, but for very large p, it can suffer from noise accumulation, as too many irrelevant predictors are being considered for prediction purposes (Mukhopadhyay and Dunson 2020). Therefore, screening out irrelevant variables before performing the random projection is advisable in order to tackle this issue. The screening can be performed in a probabilistic fashion, by randomly sampling covariates for inclusion in the model based on probabilities proportional to an importance measure (as opposed to random subspace sampling employed in e.g., random forests). Finally, in practice, the information from multiple such screening and projections can be combined by averaging, in order to reduce the variance introduced by the random sampling (of both projections and screening indicators) (Thanei, Heinze, and Meinshausen 2017). To address these points, the implemented algorithm builds an ensemble of GLMs in the spirit of Mukhopadhyay and Dunson (2020) and Parzer, Filzmoser, and Vana-Gür (2024) where, in each model of the ensemble, i) variables are first screened based on a screening coefficient, ii) the selected variables are then projected to lower dimensional space using, iii) a GLM is estimated using the projected predictors.

The package provides methods such `plot`, `predict`, `coef`, `print`, which allow users to more easily interact with the model output and analyze the results. The GLM framework, especially when combined with random projections which preserve information on the original coefficients (such as the one in Parzer *et al.* 2024), facilitates interpretability of the model output, allowing users to understand variable effects.

While **SPAR** offers the first implementation of the described algorithm and to the best of our knowledge no other package offers the same functionality for GLMs, few other R packages focus on building ensembles of *classifiers* where the dimensionality of the predictors in reduced. Most notably, package **RPEnsemble** (Cannings and Samworth 2021) implements the procedure in Cannings and Samworth (2017) where "carefully-selected" random projections are used for projecting the predictors before they are employed in a classifier such as k-nearest neighbor, linear or quadratic discriminant analysis. On the other hand, package **RaSEn** (Tian and Feng 2021) implements the RaSE algorithm for ensemble classification and regression problems, where random subspaces are generated and the optimal one is chosen to train a weak learner on the basis of some criterion.

Furthermore, several packages in R provide functionality for random projections. For instance, package **RandPro** (Aghila and Siddharth 2020; Siddharth and Aghila 2020) allows for four different random projection matrices to be applied to the predictor matrix before employing one of k–nearest neighbor, support vector machine or naive Bayes classifier non the projected features. Package **SPCAvRP** (Gataric, Wang, and Samworth 2019) implements sparse principal component analysis, based on the aggregation of eigenvector information from "carefully-selected" axis-aligned random projections of the sample covariance matrix. Additionally, package **RPEnsembleR** (Cannings and Samworth 2021) implements a similar idea when building the ensemble of classifiers: for each classifier in the ensemble, a collection of (Gaussian, axis-aligned projections, or Haar) random projection matrices is generated, and the one that minimizes a risk measure for classification on a test set is selected. For Python (van Rossum *et al.* 2011) the **sklearn.random_projection** module (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg *et al.* 2011) implements two types of unstructured random matrix, namely Gaussian random matrix and sparse random matrix (which are introduced in Section 2.2).

Regarding variable screening, users can seamlessly integrate various techniques by either writing their own procedures or leveraging the extensive array of packages available in R. An overview of some notable packages on the Comprehensive R Archive Network (CRAN) includes the following packages. Package **SIS** (Saldana and Feng 2018), which implements the (iterative) sure independence screening procedure and its extensions, as detailed in Fan and Lv (2007), Fan and Song (2010), Fan, Feng, and Wu (2010). This package also provides functionality for estimating a penalized generalized linear model or a cox regression model for the variables selected by the screening procedure. Package **VariableScreening** (Li, Huang, and Dziak 2022) offers screening methods for iid data, varying-coefficient models, and longitudinal data and includes techniques such as sure independent ranking and screening (SIRS), which ranks the predictors by their correlation with the rank-ordered response, or distance correlation sure independence screening (DC-SIS), a non-parametric extension of the correlation coefficient. Package **MFSIS** (Cheng, Wang, Zhu, Zhong, and Zhou 2024) provides a collection of model-free screening techniques including SIRS, DC-SIS, the fused Kolmogorov filter (Mai and Zou 2015) the projection correlation method using knock-off features (Wanjun Liu and Li 2022), among others. Additional packages implement specific procedures but their review is beyond the scope of the current paper.

The rest of the paper is organized as follows: Section 2 provides the methodological details of the implemented algorithm. The package is described in Section 3 and Section 4 exemplifies how a new screening coefficient and a new random projection can be integrated in the package. Section 5 contains two examples of employing the package on real data sets. Finally, Section 6 concludes.

2. Methods

The package implements a procedure for building an ensemble of GLMs where we employ screening and random projection to the predictor matrix pre-model estimation for the purpose of dimensionality reduction.

Throughout the section we assume to observe high-dimensional data $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^p$ is a predictor vector and $y_i \in \mathbb{R}$ is the response, with $p \gg n$.

2.1. Variable screening

In this section we provide an overview of possible approaches to performing variable screening in a high-dimensional setting. In high-dimensional modeling, the goal of variable screening is to reduce the predictor set by selecting a small subset of variables with a strong *utility* to the response variable. This initial selection enables more efficient downstream analyses by discarding less relevant predictors early in the modeling process, thus reducing computational costs and potential noise accumulation stemming from irrelevant variables (see e.g., Mukhopadhyay and Dunson 2020).

Classic approaches such as sure independence screening (SIS), proposed by Fan and Lv (2007), use the vector of marginal empirical correlations $\hat{\omega} = (\omega_1, \dots, \omega_p)' \in \mathbb{R}^p, \omega_j = \mathrm{Cor}(X_{.j}, y)$ to screen predictors in a linear regression setting by selecting the variable set $\mathcal{A}_\gamma = \{j \in [p] : |w_j| > \gamma\}$ depending on a threshold $\gamma > 0$, where $[p] = \{1, \dots, p\}$. Under certain technical conditions, this screening coefficient has the *sure screening property* $\mathbb{P}(\mathcal{A} \subset \mathcal{A}_{\gamma_n}) \to 1$ for $n \to \infty$. Extensions to SIS include modifications for GLMs (Fan and Song 2010), where screening is performed based on the log-likelihood or the slope coefficient of the GLM containing only X_j as a predictor: $\hat{\omega}_j =: \text{argmin}_{\beta \in \mathbb{R}} \text{min}_{\beta_0 \in \mathbb{R}} \sum_{i=1}^n -\ell(\beta_j, \beta_0; y_i, x_{ij})$. However, both approaches face limitations related to the required technical conditions which can rule out practically possible scenarios where an important variable is marginally uncorrelated to the response due to their multicollinearity. To tackle these issues, Fan, Samworth, and Wu (2009) propose to use an iterative procedure where SIS is applied subsequently on the residuals of the model estimated in a previous step. Additionally, in a linear regression setting, Wang and Leng (2016) propose employing the ridge estimator when the penalty term converges to zero while Cho and Fryzlewicz (2012) propose using the tilted correlation, i.e., the correlation of a tilted version of X_j with y where the effect of other variables is reduced. For discrete outcomes, joint feature screening (Xu and Chen 2014) has been proposed.

In order to tackle potential model misspecification, a rich stream of literature focuses on developing semi- or non-parametric alternatives to SIS. For linear regression, approaches include using the ranked correlation (Zhu, Li, Li, and Zhu 2011), (conditional) distance correlation (Li, Zhong, and Zhu 2012; Wang, Pan, Hu, Tian, and Zhang 2015) or quantile correlation (Ma and Zhang 2016). For GLMs, Fan, Feng, and Song (2011) extend Fan and Song (2010) by fitting a generalized additive model with B-splines. Further extensions for discrete (or categorical) outcomes include the fused Kolmogorov filter (Mai and Zou 2013), the mean conditional variance, i.e., the expectation in X_j of the variance in the response of the conditional cumulative distribution function $\mathsf{P}(X \leq x|Y)$ (Cui, Li, and Zhong 2015). Ke (2023) propose a model free method where the contribution of each individual predictor is quantified marginally and conditionally in the presence of the control variables as well as the other candidates by reproducing-kernel-based R^2 and partial R^2 statistics.

SPAR allows the integration of various advanced screening techniques using a flexible framework, which in turn enables users to apply various screening methods tailored to their data characteristics in the algorithm generating the ensemble. This flexibility allows users to evaluate different strategies, ensuring that the most effective approach is chosen for the specific application at hand. Moreover, it incorporates probabilistic screening strategies, which can be particularly useful in ensembles, as they enhance the diversity of predictors across ensemble models. Instead of relying on a fixed threshold or number of predictors to be screened, predictors are sampled with probabilities proportional to their screening score (see Mukhopadhyay and Dunson 2020; Parzer *et al.* 2024).

2.2. Random projection tools

SPAR has been designed to allow the incorporation of various random projection techniques, enabling users to tailor the procedure to their specific data needs. Below, we provide background information on random projection techniques and an overview of possible choices for building such random projection matrices.

The random projection method relies on the Johnson-Lindenstrauss (JL) lemma (Johnson and Lindenstrauss 1984), which asserts that there exists a linear map $\Phi \in \mathbb{R}^{m \times p}$ that embeds any set of points in p-dimensional Euclidean space collected in the rows of $X \in \mathbb{R}^{n\times p}$ into a m-dimensional Euclidean space where pairwise distances are approximately preserved within a factor of $1 \pm \epsilon)$ for $m \leq m_0 = \mathcal{O}(\epsilon^{-2}\log(n))$. Computationally, an attractive feature of the method for high-dimensional settings is that the bound does not depend on p.

The random map Φ that satisfies the JL lemma with high probability should be chosen such that it fulfills certain conditions (see Johnson and Lindenstrauss 1984) and the literature focuses on constructing such matrices either by sampling them from some "appropriate" distribution, by inducing sparsity in the matrix and/or by employing specific fast constructs which lead to efficient matrix-vector multiplications. It turns out that the conditions are generally satisfied by nearly all sub-Gaussian distributions (Matoušek 2008). A common choice is the standard normal distribution $\Phi_{ij} \overset{iid}{\sim} N(0,1)$ (Frankl and Maehara 1988) or a sparser version $\Phi_{ij} {\sim} N(0, 1/\sqrt{\psi})$ with probability ψ and 0 otherwise (Matoušek 2008). Another computationally simpler option is the Rademacher distribution where $\Phi_{ij} = \pm 1/\sqrt{\psi}$ with probability $\psi/2$ and zero otherwise for $0 < \psi \leq 1$, where Achlioptas (2003) shows results for $\psi = 1$ and $\psi = 1/3$ while Li, Hastie, and Church (2006) recommend using $\psi = 1/\sqrt{p}$ to obtain very sparse matrices.

Further approaches include using Haar measure to generate random orthogonal matrices (Cannings and Samworth 2017) or non-sub-Gaussian distribution like the standard Cauchy, proposed by **?** for preserving approximate ℓ_1 distances in settings where the data is high-dimensional, non-sparse, and heavy-tailed. Structured matrices, which allow for more efficient multiplication, have also been proposed (see e.g., Ailon and Chazelle 2009; Clarkson and Woodruff 2013).

The conventional random projections mentioned above are data-agnostic. However, recent work has proposed incorporating information from the data either to select the "best" random projection or to directly inform the random projection procedure. For example, Cannings and Samworth (2017) build an ensemble classifier where the random projection matrix is chosen by selecting the one that minimizes the test error of the classification problem among a set of data-agnostic random projections.

On the other hand, Parzer *et al.* (2024) propose to use a random projection matrix for GLMs which directly incorporates information about the relationship between the predictors and the response in the projection matrix, rather than a projection matrix which satisfies the JL lemma. This approach constructs a projection matrix using the sparse embedding matrix of Clarkson and Woodruff (2013), where the random diagonal elements are replaced by a ridge coefficient with a minimal λ penalty. This method has the advantage of allowing the regression coefficients to be recovered after the projection, which is in general not possible with conventional random projections (Thanei *et al.* 2017).

Moreover, another data-driven approach to random projection for regression has been proposed by Ryder, Karnin, and Liberty (2019), who propose a data-informed random projection using an asymmetric transformation of the predictor matrix without using information of the response.

2.3. Generalized linear models

After we perform in each marginal model an initial screening step followed by a projection step, we assume that the reduced and projected set of predictors z_i together with the response arises from a GLM with the response having conditional density from a (reproductive) exponential dispersion family of the form

$$f(y_i|\theta_i, \phi) = \exp\Big\{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)\Big\}, \quad g(\mathsf{E}[y_i|z_i]) = \gamma_0 + z_i'\gamma =: \eta_i, \tag{1}$$

where θ_i is the natural parameter, $a(.) > 0$ and $c(.)$ are specific real-valued functions determining different families, ϕ is a dispersion parameter, and $b(.)$ is the log-partition function normalizing the density to integrate to one. If ϕ is known, we obtain densities in the natural exponential family for our responses. The responses are related to the m-dimensional reduced and projected predictors through the conditional mean, i.e., the conditional mean of y_i given z_i depends on a linear combination of the predictors through a (invertible) link function $g(.)$, where $\gamma_0 \in \mathbb{R}$ is the intercept and $\gamma \in \mathbb{R}^m$ is a vector of regression coefficients for the m projected predictors.

Given that m, the goal dimension of the projection need not necessarily be small in comparison to n (we recommend using a dimension of around $n/2$), we observe that adding a small L_2 penalty in the marginal models, especially for the binomial family, can make estimation more stable as it alleviates problems related to separation. The marginal models we estimate therefore involve minimizing the following function:

$$\text{argmin}_{\gamma\in\mathbb{R}^m} \min_{\gamma_0\in\mathbb{R}} \sum_{i=1}^{n} -\ell(\gamma_0, \gamma; y_i, z_i) + \frac{\varepsilon}{2}\sum_{j=1}^{m}\gamma_j^2,\ \varepsilon > 0.$$

We will employ for this purpose function `glmnet()` of package glmnet (Tay, Narasimhan, and Hastie 2023) in the estimation of the marginal models.

2.4. Algorithm

We present the general algorithm implemented in package **SPAR**.

1. Choose family with corresponding log-likelihood $\ell(.)$ and link
2. Standardize predictors $X : n \times p$ by subtracting the sample column mean and dividing by the sample standard deviation.
3. Calculate screening coefficients $\hat{\omega}$.
4. For $k = 1, \dots, M$ models:

4.1. screen $2n$ predictors based on the screening coefficient $\hat{\omega}_j$ yielding screening index set $I_k = \{j_1^k, \dots, j_{2n}^k\} \subset [p]$; if probabilistic screening should be employed draw the predictors sequentially without replacement using an initial vector of probabilities $p_j \propto |\hat{\omega}_j|$. Otherwise, select the $2n$ variables with the highest $|\hat{\omega}_j|$.

4.2. project screened variables to a random dimension $m_k \sim \text{Unif}\{\log(p), \dots, n/2\}$ using **projection matrix** Φ_k to obtain $Z_k = X_{\cdot I_k}\Phi_k^\top \in \mathbb{R}^{n\times m_k}$:

4.3. fit a (L_2 penalized) **GLM** of y against Z_k to obtain estimated coefficients $\hat{\gamma}^k \in \mathbb{R}^{m_k}$ and $\hat{\beta}^k_{I_k} = \Phi_k'\hat{\gamma}^k$, $\hat{\beta}^k_{\bar{I}_k} = 0$.

5. For a given threshold $\nu > 0$, set all $\hat{\beta}_j^k$ with $|\hat{\beta}_j^k| < \nu$ to 0 for all j, k.
6. Choose M and ν via a validation set or cross-validation by repeating steps 1 to 4. Choose

$$(M_{\text{best}}, \nu_{\text{best}}) = \text{argmin}_{M,\nu}\text{Dev}(M, \nu) \tag{2}$$

7. Combine models of the ensembles via the coefficients using **simple average** $\hat{\beta} = \sum_{k=1}^{M} \hat{\beta}^k / M$.
8. Output the estimated coefficients and predictions for the chosen M and ν.

3. Software

The package can be installed from GitHub

```
R> devtools::install_github("RomanParzer/SPAR")
```

and loaded by

```
R> library("SPAR")
```

In this section we rely for illustration purposes on the example data set from the package which has a continuous response `y` and 2000 predictors `x`:

```
R> data("example_data", package = "SPAR")
R> str(example_data)
#> List of 7
#>  $ x     : num [1:200, 1:2000] 1.8302 -0.4251 -1.3893 -0.0947 0.4304 ...
#>  $ y     : num [1:200] -5.64 -23.63 -17.09 13.18 20.91 ...
#>  $ xtest : num [1:100, 1:2000] -0.166 -0.3729 0.0379 0.6774 0.2174 ...
#>  $ ytest : num [1:100] 10.61 -34.1 29.3 35.53 8.67 ...
#>  $ mu    : num 1
#>  $ beta  : num [1:2000] 1 -2 3 2 1 -3 2 3 1 -2 ...
#>  $ sigma2: num 83
```

3.1. Main functions and their arguments

The two main functions for fitting the SPAR algorithm are:

```
R> spar(x, y, family = gaussian("identity"), rp = NULL, screencoef = NULL,
+   xval = NULL, yval = NULL, nnu = 20, nus = NULL, nummods = c(20),
+   measure = c("deviance", "mse", "mae", "class", "1-auc"),
+   inds = NULL, RPMs = NULL, ...)
```

which implements the algorithm in Section 2.4 without cross-validation and returns an object of class '`spar`', and

```
R> spar.cv(x, y, family = gaussian("identity"), rp = NULL, screencoef = NULL,
+   nfolds = 10, nnu = 20, nus = NULL, nummods = c(20),
+   measure = c("deviance", "mse", "mae", "class", "1-auc"), ...)
```

which implements the cross-validated procedure and returns an object of class '`spar.cv`'.

The common arguments of these functions are:

- `x` an $n \times p$ numeric matrix of predictor variables,
- `y` numeric response vector of length n,
- `family` object from `stats::family()`; defaults to `gaussian()`;
- `rp` an object of class '`randomprojection`', defaults to `rp_cw(data=TRUE)`;
- `screencoef` an object of class '`screencoef`', defaults to `screen_glmnet()`;
- `nnu` is the number of threshold values ν which should be considered for thresholding; defaults to 20;
- `nus` is an optional vector of ν values to be considered for thresholding. If it is not provided, is defaults to a grid of `nnu` values. This grid is generated by including zero and `nnu`-1 quantiles of the absolute values of the estimated coefficients from the marginal models, chosen to be equally spaced on the probability scale .
- `nummods` is the number of models to be considered in the ensemble; defaults to 20. If a vector is provided, all combinations of `nus` and `nummods` are considered when choosing the optimal ν_{best} and M_{best}.
- `measure` gives the measure based on which the thresholding value ν_{opt} and the number of models `M` should be chosen on the validation set (for `spar()`) or in each of the folds (in `spar.cv()`). Defaults to `"deviance"`, which is available for all families. Other options are `"mse"` or `"mae"` (between responses and predicted means, for all families), `"class"` (misclassification error) and `"1-auc"` (one minus area under the ROC curve) both just for binomial family.

Furthermore, `spar()` has the specific arguments:

- `xval` and `yval` which are used as validation sets for choosing ν_{best} and M_{best}. If not provided, `x` and `y` will be employed.
- `inds` is an optional list of length `max(nummods)` containing column index-vectors corresponding to variables that should be kept after screening for each marginal model; dimensions need to fit those of the dimensions of the provided matrices in `RPM`.
- `RPMs` is an optional list of length `max(nummods)` which contains projection matrices to be used in each marginal model.

Function `spar.cv()` has the specific argument `nfolds` which is the number of folds to be used for cross-validation. It relies on `spar()` as a workhorse, which is called for each fold. The random projections for each model are held fixed throughout the cross-validation to reduce the computational burden. This is possible by calling `spar()` in each fold with a predefined `inds` and `RPMs` argument, which are generated by first calling `spar()` on the whole dataset, before starting the cross-validation procedure. However, it is possible to specify whether the data associated with the random projection (relevant for data-driven random projections) should be updated in each fold iteration with the corresponding training data. This is achieved by modifying elements of the '`randomprojection`' object, which we will exemplify in Section 4.

3.2. Screening coefficients

The objects for creating screening coefficients are implemented as S3 classes '`screencoef`'. These objects are created by several implemented `screen_*` functions, which take `...` and a list of controls `control` as arguments.

Consider as an example function `screen_marglik` which implements a screening procedure based on the coefficients of univariate GLMs:

```
R> screen_marglik
#> function(..., control = list()) {
#>     out <- list(name = name,
#>                 generate_fun = generate_fun,
#>                 control = control)
#>     attr <- list2(...)
#>     attributes(out) <- c(attributes(out), attr)
#>     if (is.null(attr(out, "type"))) {
#>       attr(out, "type") <- "prob"
#>     } else {
#>       stopifnot(
#>         "'type' must be either 'prob' or 'fixed'." =
#>           (attr(out, "type") == "prob" | attr(out, "type") == "fixed")
#>       )
#>     }
#>     class(out) <- c("screencoef")
#>     return(out)
#>   }
#> <bytecode: 0x128e74968>
#> <environment: 0x128e69960>
```

Arguments related to the screening procedure can be passed through `...`, and will be saved as attributes of the '`screencoef`' object. More specifically, the following attributes are relevant for function `spar()`:

- `nscreen` integer giving the number of variables to be retained after screening; defaults to $2n$.
- `split_data_prop`, double between 0 and 1 which indicates the proportion of the data that should be used for computing the screening coefficient. The remaining data will be used for estimating the marginal models in the SPAR algorithm; defaults to `NULL`. In this case the whole data will be used for estimating the screening coefficient and the maginal models.
- `type` character – either `"prob"` (indicating that probabilistic screening should be employed) or `"fixed"` (indicating that a fixed set of `nscreen` variables should be employed across the ensemble; defaults to `type = "prob"`.

The `control` argument, on the other hand, is a list containing extra parameters to be passed to the main function computing the screening coefficients.

The following screening coefficients are implemented in **SPAR**:

- `screen_marglik()` - computes the screening coefficients by the coefficient of x_j for $j = 1, \dots, p$ in a univariate GLM using the `stats::glm()` function.

$$\hat{\omega}_j =: \text{argmin}_{\beta_j \in \mathbb{R}} \text{min}_{\beta_0 \in \mathbb{R}} \sum_{i=1}^{n} -\ell(\beta_0, \beta_j; y_i, x_{ij})$$

  It allows to pass a list of controls through the `control` argument to `stats::glm` such as weights, family, offsets.

- `screen_cor()` – computes the screening coefficients by the correlation between y and x_j using the function `stats::cor()`. It allows to pass a list of controls through the `control` argument to `stats::cor`.
- `screen_glmnet()` – computes by default the ridge coefficient where the penalty λ is very small (see Parzer *et al.* 2024, for clarification).

$$\hat{\omega} =: \text{argmin}_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} -\ell(\beta; y_i, x_i) + \frac{\varepsilon}{2} \sum_{j=1}^{p} \beta_j^2,\ \varepsilon > 0$$

  The function relies on `glmnet::glmnet()` and, while it assumes by default $\alpha = 0$ and a small penalty, it allows to pass a list of controls through the `control` argument to `glmnet::glmnet()` such as `alpha = 1`. Used as default if `screencoef = NULL` in function call of `spar()` or `spar.cv()`.

All implemented `screen_*` functions return an object of class '`screencoef`' which in turn is a list with three elements:

- a character `name`,
- `generate_fun` – an R function for generating the screening coefficient. This function should have as following arguments the '`screencoef`' object itself and `data`, which should be a list of `x` – the matrix of standardized predictors – and `y` – the vector of (standardized in the Gaussian case) responses. It returns a vector of screening coefficients of length p.
- `control`, which is the control list passed by the user in `screen_*`. These controls are arguments which are needed in `generate_fun` in order to generate the desired screening coefficients.

For illustration purposes, consider the object created by calling `screen_marglik()`:

```
R> obj <- screen_marglik()
```

A user-friendly `print` of the '`screencoef`' is provided:

```
R> obj
#> Name: screen_marglik
#> Main attributes:
#> * proportion of data used for screening: 1
#> * number of screened variables: not provided, will default to 2n
#> * type: probabilistic screening
#> * screening coefficients: not yet computed from the data.
```

The structure of the object is the following:

```
R> unclass(obj)
#> $name
#> [1] "screen_marglik"
#>
#> $generate_fun
#> function(object, data) {
#>   y <- data$y
#>   x <- data$x
#>   if (is.null(object$control$family)) {
#>     object$control$family <- attr(object, "family")
#>   }
#>   coefs <- apply(x, 2, function(xj){
#>     glm_res <- do.call(function(...) glm(y ~ xj,  ...),
#>                        object$control)
#>     glm_res$coefficients[2]
#>   })
#>   coefs
#> }
#> <environment: namespace:SPAR>
#>
#> $control
#> list()
#>
#> attr(,"type")
#> [1] "prob"
```

Function `generate_fun` defines the generation of the screening coefficient. Note that it considers the controls in `object$control` when calling the `stats::glm()` function (unless it is provided, the `family` argument in `stats::glm()` will be set to the "global" family of the SPAR algorithm which is assigned inside the `spar()` function an attribute for the `screencoef` object).

For convenience, a constructor function `constructor_screencoef()` is provided, which can be used to create new `screen_*` functions. An example is presented in Section 4.1.

3.3. Random projections

Similar to the screening procedure, the objects for creating random projections are implemented as S3 classes '`randomprojection`' and are created by functions `rp_*(..., control = list())`, which take `...` and a list of controls `control` as arguments.

Arguments related to the random projection can be passed through `...`, which will then be saved as attributes of the '`randomprojection`' object. More specifically, the following attributes are relevant in the SPAR algorithm:

- `mslow`: integer giving the minimum dimension to which the predictors should be projected; defaults to $\log(p)$.

- `msup`: integer giving the maximum dimension to which the predictors should be projected; defaults to $n/2$.
- `data`: boolean indicating whether the random projection is data-driven.

Note that for random projection matrices which satisfy the JL lemma, `mslow` can be determined by employing existing results which give a lower bound on the goal dimension in order to preserve the distances between all pairs of points within a factor $(1 \pm \epsilon)$. For example, **?** show $m_0 = \log n(4 + 2\tau)/(\epsilon^2/2 - \epsilon^3/3)$ for probability $1 - n^{-\tau}$.

The following random projections are implemented in **SPAR**:

- `rp_gaussian()` – random projection object where the generated matrix will have iid entries from a normal distribution (defaults to standard normal entries)
- `rp_sparse()` – random projection object where the generated matrix will be the one in (Achlioptas 2003) with `psi=1` by default.
- `rp_cw()` – sparse embedding random projection in (Clarkson and Woodruff 2013) for `rp_cw(data = FALSE)`. Defaults to `rp_cw(data=TRUE)`, which replaces the random elements on the diagonal by the ridge coefficients with a small penalty, as introduced in Parzer *et al.* (2024).

The `rp_*` functions return an object of class '`randomprojection`' which is a list with three elements:

- `name`,
- `generate_fun` function for generating the random projection matrix. This function should have arguments `rp`, which is itself a '`randomprojection`' object, `m`, the target dimension and a vector of indices `included_vector` which indicates the column index of the original variables in the `x` matrix to be projected using the random projection. This is needed due to the fact that screening can be employed pre-projection. It can return a matrix or a sparse matrix of class "`dgCMatrix`" of the **Matrix** with `m` rows and `length(included_vector)` columns.
- `update_data_rp` optional function used for data-driven random projections, which updates the '`randomprojection`' object with data information which is relevant for the random projection. The updating happens only once, before the start of the SPAR algorithm, where appropriate attributes are added to the '`randomprojection`' object. All relevant quantities are to be used in the `generate_fun` function. This function should have arguments `rp`, which is a '`randomprojection`' object to be updated and `data`, which should be a list of `x` – the matrix of standardized predictors – and `y` – the vector of (standardized in the Gaussian case) responses. Returns a `randomprojection` object.
- `update_rpm_w_data` optional function for updating the random projection matrices provided in the argument `RPMs` of functions `spar` and `spar.cv` with data-dependent parameters. While `update_data_rp` is employed only once at the start of the algorithm, `update_rpm_w_data` specifies how to modify each random projection provided in `RPMs`. This is particularly relevant for the cross-validation procedure, which employs the random projection matrices generated by calling the `spar()` function on the whole dataset

before starting the cross-validation exercise.For example, in our implementation of the data-driven `rp_cw()`, we only update `RPMs` by adjusting with the vector of screening coefficients computed on the current training data in each fold, but do not modify the random elements in each fold, to reduce the computational burden. Defaults to `NULL`. If not provided, the values of the provided `RPMs` do not change.

- `control`, which is the control list in `rp_*`. These controls are arguments needed in `generate_fun` in order to generate the desired random projection.

For illustration purposes, consider the implemented function `rp_gaussian`, which is a random projection with entries drawn from the normal distribution. The `print` method returns key information about the random projection procedure.

```
R> obj <- rp_gaussian()
R> obj
#> Name: rp_gaussian
#> Main attributes:
#> * Data-dependent: FALSE
#> * Lower bound on goal dimension m: not provided, will default to log(p).
#> * Upper bound on goal dimension m: not provided, will default to n/2.
```

We turn to looking at the structure of the object:

```
R> unclass(obj)
#> $name
#> [1] "rp_gaussian"
#>
#> $generate_fun
#> function(rp, m, included_vector) {
#>   p <- length(included_vector)
#>   control_rnorm <-
#>     rp$control[names(rp$control)  %in% names(formals(rnorm))]
#>   vals <- do.call(function(...)
#>     rnorm(m * p, ...), control_rnorm)
#>   RM <- matrix(vals, nrow = m, ncol = p)
#>   return(RM)
#> }
#> <environment: namespace:SPAR>
#>
#> $update_data_fun
#> NULL
#>
#> $update_rpm_w_data
#> NULL
#>
#> $control
#> list()
#>
#> attr(,"data")
#> [1] FALSE
```

The `generate_fun` returns a matrix with `m` rows and `length(included_vector)` columns. Note that `included_vector` gives the indices of the variables which have been selected by the screening procedure. In this case, where the random projection does not use any data information, we are only interested in the length of this vector.

The element `update_data_fun` and `update_rpm_w_data` are `NULL` as this conventional random projection is data-agnostic.

3.4. Methods

Methods `print`, `plot`, `coef`, `predict` are available for both '`spar`' and '`spar.cv`' classes.

print

The `print` method returns information on ν_{best} M_{best}, the number of active predictors (i.e., predictors which have at least a nonzero coefficient across the marginal models) and a five-point summary of the non-zero coefficients.

```
R> spar_res <- spar(example_data$x, example_data$y,
+                 xval = example_data$xtest,
+                 yval = example_data$ytest,
+                 nummods = c(5,10,15,20,25,30))
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.476575e-07
R> spar_cv <- spar.cv(example_data$x, example_data$y,
+                   nummods = c(5,10,15,20,25,30))
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.476575e-07
```

```
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.469866e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.409689e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.415077e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.476504e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.453844e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.4416e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.482035e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
```

```
#> $lambda.min.ratio
#> [1] 1.434129e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.46346e-07
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 1.441134e-07
```

```
R> spar_res
#> SPAR object:
#> Smallest Validation Measure reached for nummod=5,
#>               nu=1.53e-02 leading to 592 / 2000 active predictors.
#> Summary of those non-zero coefficients:
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -1.02824 -0.15215  0.10434  0.05272  0.22101  1.93862
```

```
R> spar_cv
#> SPAR.cv object:
#> Smallest CV-Meas 2074.5 reached for nummod=20,
#>               nu=0.00e+00 leading to 1745 / 2000 active predictors.
#> Summary of those non-zero coefficients:
#>      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
#> -0.787619 -0.044052  0.006825  0.019401  0.079574  1.086143
#>
#> Sparsest coefficient within one standard error of best CV-Meas
#>               reached for nummod=5, nu=6.86e-03
#> leading to 933 / 2000 active
#>               predictors with CV-Meas 2681.5.
#> Summary of those non-zero coefficients:
```

```
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -1.26330 -0.12629  0.05800  0.03578  0.17439  1.64770
```

coef

Method `coef` takes as inputs a `spar` or `spar.cv` object, together with further arguments:

- `nummod` – number of models used to compute the averaged coefficients; value of `nummod` with minimal `measure` is used if not provided.
- `nu` – threshold level used to compute the averaged coefficients; value with minimal `measure` is used if not provided.

```
R> str(coef(spar_res))
#> List of 4
#>  $ intercept: num 2.58
#>  $ beta     : num [1:2000] 0.0907 0 0.7089 0 0 ...
#>  $ nummod   : num 5
#>  $ nu       : num 0.0153
```

It returns a list with the intercept, vector of `beta` coefficients and the `nummod` and `nu` employed in the calculation.

Additionally for ‘`spar.cv`’, the `coef` method also has argument `opt_par` which is one of `c("1se","best")` and chooses whether to select the best pair of `nus` and `nummods` according to cross-validation `measure`, or the solution yielding sparsest vector of coefficients within one standard deviation of that optimal cross-validation `measure`. This argument is ignored when `nummod` and `nu` are given.

predict

Functionality for computing predictions is provided through the method `predict` which takes a `spar` or `spar.cv` object, together with

- `xnew` – matrix of new predictor variables; must have same number of columns as `x`.
- `type` – the type of required predictions; either on `"response"` level (default) or on `"link"` level
- `avg_type` – type of averaging used across the marginal models; either on `"link"` (default) or on `"response"` level
- `nummod` – number of models used to compute the averaged coefficients; value of `nummod` with minimal `measure` is used if not provided.
- `nu` – threshold level used to compute the averaged coefficients; value with minimal `measure` is used if not provided.
- `coef` – optional vector of coefficients to be used directly in the prediction.

Additionally, for class ‘`spar.cv`’, argument `opt_par` is available and used in the computation of the coefficients to be used for prediction (see above description of method `coef`).

plot

Plotting functionality is provided through the `plot` method, which takes a `spar` or `spar.cv` object, together with further arguments:

- `plot_type` – one of:
  - `"Val_Measure"` plots the (cross-)validation `measure` for either a grid of `nu` values for a fixed number of models `nummod` or viceversa.
  - `"Val_numAct"` plots the number of active variables for either a grid of `nu` values for a fixed number of models `nummod` or viceversa.
  - `"res-vs-fitted"` produces a residuals-vs-fitted plot. The residuals are computed as $y - \hat{y}$, where $\hat{y}$ is the prediction computed on response level.
  - `"coefs"` produces a plot of the value of the standardized coefficients for each predictor in each marginal model (before thresholding). For each predictor, the values of the coefficients are sorted from largest to smallest across the marginal models and then represented in the plot.
- `plot_along` – one of `c("nu","nummod")`; for `plot_type = "Val_Measure"` or `plot_type = "Val_numAct"` it indicates whether the values of the cross-validation measure or number of active variables, respectively, should be shown for a grid of ν values while keeping the number of models `nummod` fixed or viceversa. This argument is ignored when `plot_type = "res-vs-fitted"` or `plot_type = "coefs"`.
- `nummod` – fixed value for number of models when `plot_along = "nu"` for `plot_type = "Val_Measure"` or `"Val_numAct"`; if `plot_type = "res-vs-fitted"`, it is used in the `predict` method, as described above.
- `nu` – fixed value for ν when `plot_along = "nummod"` for `plot_type = "Val_Measure"` or `"Val_numAct"`; if `plot_type = "res-vs-fitted"`, it is used in the `predict` method, as described above.
- `xfit` – if `plot_type = "res-vs-fitted"`, it is the matrix of predictors used in computing the fitted values. Must provided for this plot, as the `spar` or `spar.cv` objects do not store the original data.
- `yfit` – if `plot_type = "res-vs-fitted"`, vector of responses used in computing the residuals. Must provided for this plot, as the `spar` or `spar.cv` objects do not store the original data.
- `prange` – optional vector of length 2 in case `plot_type = "coefs"` which gives the limits of the predictors' plot range; defaults to `c(1, p)`.
- `coef_order` – optional index vector of length p in case `plot_type = "coefs"` to give the order of the predictors; defaults to `1 : p`.

For class '`spar.cv`' there is the extra argument `opt_par = c("best", "1se")` which, for `plot_type = "res-vs-fitted"` indicates whether the predictions should be based on coefficients using the best (ν, M) combination or on the one which delivers the sparsest β having validation measure within one standard deviation from the minimum

The `plot` methods return objects of class “`ggplot`” (Wickham 2016).

4. Extensibility

4.1. Screening coefficients

We exemplify how new screening coefficients implemented in package **VariableScreening** can easily be used in the framework of **SPAR**.

We start by defining the function for generating the screening coefficients using the `screenIID()` function in **VariableScreening**.

```
R> generate_scr_sirs <- function(object, data) {
+  res_screen <- do.call(function(...)
+    VariableScreening::screenIID(data$x, data$y, ...),
+    object$control)
+  coefs <- res_screen$measurement
+  coefs
+}
```

Note that `screenIID()` also takes method as an argument. To allow for flexibility, we do not fix the method in `generate_scr_sirs` but rather allow the user to pass a method through the `control` argument in the `screen_*` function. This function is created using `constructor_screencoef`:

```
R> screen_sirs <- constructor_screencoef(
+  "screen_sirs",
+  generate_fun = generate_scr_sirs)
```

We now call the `spar()` function with the newly created screening procedure. We consider the method SIRS of Zhu *et al.* (2011), which ranks the predictors by their correlation with the rank-ordered response and we do not perform probabilistic variable screening but employ the top $2n$ variables in each marginal model.

```
R> set.seed(123)
R> spar_example <- spar(example_data$x, example_data$y,
+    screencoef = screen_sirs(type = "fixed",
+      control=list(method = "SIRS")),
+                   measure = "mse")
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#>
```

```
#> $lambda.min.ratio
#> [1] 1.476575e-07
R> spar_example
#> SPAR object:
#> Smallest Validation Measure reached for nummod=20,
#>               nu=1.75e-03 leading to 396 / 2000 active predictors.
#> Summary of those non-zero coefficients:
#>       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
#> -0.6918230 -0.0928444  0.0009116  0.0943062  0.1777889  2.0089709
```

4.2. Random projections

We exemplify how new random projections can be implemented in the framework of **SPAR**.

We implement the random projection of Cannings and Samworth (2017), who propose using the Haar measure for generating the random projections. They simulate matrices from the Haar measure by independently drawing each entry of a matrix Q from a standard normal distribution, and then to take the projection matrix to be the transpose of the matrix of left singular vectors in the singular value decomposition of Q. Moreover, they suggest using "good" random projections, in the sense that they deliver the best out-of-sample prediction. The proposed approach employs B_1 models in an ensemble of classifiers and for each model k, B_2 data independent random projections are generated and the one with the lowest error on a test set is the one chosen to project the variables in model k.

We can implement such a random projection in **SPAR** by the following building block:

```
R> update_data_cannings <- function(rp, data) {
+  attr(rp, "data") <- data
+  rp
+}
```

This is the function which adds data information to the random projection object. Here, the whole data can be added as information for the M random projection (alternatively, one could only pass sufficient statistics for computing the desired measures).

While the B_2 random projections are data agnostic, the `generate_fun` element of the random projection will need the data information in order to evaluate which method performs best in terms of an error measure. We will, in the following, define the function for the generation of the random projection matrix to be used in a model k.

This helper simulates $m \times p$ matrices from the Haar measure:

```
R> simulate_haar <- function(m, p) {
+    R0 <- matrix(1/sqrt(p) * rnorm(p * m), nrow = p, ncol = m)
+    RM <- qr.Q(qr(R0))[, seq_len(m)]
+    RM <- Matrix(t(RM), sparse = TRUE)
+}
```

The function that generates the random projection matrix for model k uses 25% of the data as a test set for choosing the best among B_2 random projections in terms of minimizing misclassification error for the binomial family and MSE for all other families:

```
R> generate_cannings <- function(rp, m, included_vector) {
+    p <- length(included_vector)
+    if (is.null(rp$control$B2)) rp$control$B2 <- 50
+    x <- attr(rp, "data")$x[, included_vector]
+    y <- attr(rp, "data")$y
+
+    B2 <- rp$control$B2
+    n <- nrow(x)
+    id_test <- sample(n, size = n %/% 4)
+    xtrain <- x[-id_test, ];  xtest <- x[id_test,]
+    ytrain <- y[-id_test];  ytest <- y[id_test]
+
+    if (is.null(rp$control$family)) rp$control$family <- attr(rp, "family")
+
+    family <- rp$control$family
+    control_glm <-
+      rp$control[names(rp$control) %in% names(formals(glm.fit))]
+
+    error_all <- lapply(seq_len(B2), FUN = function(s){
+      RM <- simulate_haar(m, p)
+      xrp <- tcrossprod(xtrain, RM)
+      mod <- do.call(function(...)
+       glm.fit(x =  cbind(1, xrp), y = ytrain, ...), control_glm)
+      eta_test <- drop(cbind(1, tcrossprod(xtest, RM)) %*% mod$coefficients)
+      pred <- family$linkinv(eta_test)
+      out <-  ifelse(family$family == "binomial",
+                     mean(((pred > 0.5) + 0) != ytest),
+                     mean((pred - ytest)^2))
+      list(RM, out)
+   })
+   id_best <- which.min(sapply(error_all, "[[", 2))
+   RM <- error_all[[id_best]][[1]]
+   return(RM)
+}
```

In the cross-validation procedure, we do not generate new matrices for each step to keep computational costs low, so we do not specify a function `update_rpm_w_data`.

Putting it all together, we get:

```
R> rp_cannings <- constructor_randomprojection(
+  "rp_cannings",
+  generate_fun = generate_cannings,
+  update_data_fun = update_data_cannings
+)
```

We can now estimate SPAR for a binomial model, where we transform the response to a binary variable.

```
R> ystar <- (example_data$y > 0) + 0
R> ystarval <- (example_data$ytest > 0) + 0
```

We use 100 models (which is in line to recommendations for B_1 in Cannings and Samworth (2017)), and no screening procedure. If no screening is desired, `nscreen` can be set to `p` in the `screen_*` function:

```
R> set.seed(123)
R> spar_example_1 <- spar(
+  x = example_data$x, y=ystar,
+  xval = example_data$xtest, yval = ystarval,
+  family = binomial(),
+  nummods = 100,
+  screencoef = screen_marglik(nscreen = ncol(example_data$x)),
+  rp = rp_cannings(control = list(B2 = 50)),
+  measure = "class"
+)
```

Using the data-driven `rp_cw()`:

```
R> spar_example_2 <- spar(x = example_data$x, y = ystar,
+  family = binomial(),
+  screencoef = screen_marglik(nscreen = ncol(example_data$x)),
+  rp = rp_cw(data = TRUE),
+  nummods = 100,
+  xval = example_data$xtest, yval = ystarval,
+  measure = "class"
+)
```

We can extract the measures on the validation set by:

```
R> head(spar_example_1$val_res)
#>   nnu          nu nummod numAct Meas
#> 1   1 0.000000000    100   2000 0.19
#> 2   2 0.002504236    100   2000 0.18
#> 3   3 0.005091038    100   2000 0.19
#> 4   4 0.007674062    100   2000 0.19
#> 5   5 0.010337517    100   2000 0.19
#> 6   6 0.013009198    100   2000 0.19
```

We can compare the two approaches by looking at the minimum `Meas` achieved:

```
R> min_val_1 <- min(spar_example_1$val_res$Meas)
R> id_best_1 <- max(which(spar_example_1$val_res$Meas == min_val_1))
R> min_val_2 <- min(spar_example_2$val_res$Meas)
R> id_best_2 <- max(which(spar_example_2$val_res$Meas == min_val_2))
R> spar_example_1$val_res[id_best_1, ]
#>    nnu         nu nummod numAct Meas
#> 11  11 0.02840239    100   2000 0.18
```

```
R> spar_example_2$val_res[id_best_2, ]
#>    nnu         nu nummod numAct Meas
#> 10  10 0.01408872    100   1612 0.16
```

5. Illustrations

5.1. Face image data

We illustrate the functionality of **SPAR** on the Isomap data set containing $n = 698$ black and white face images of size $p = 64 \times 64 = 4096$ together with the faces' horizontal looking direction angle as the response variable.[1]

```
R> url1 <- "https://web.archive.org/web/20150922051706/"
R> url2 <- "http://isomap.stanford.edu/face_data.mat.Z"
R> download.file(paste0(url1, url2),
+              file.path("face_data.mat.Z"))
R> system('uncompress face_data.mat.Z')
```

The `.mat` file format can be read using **R.matlab** (Bengtsson 2022)

```
R> library("R.matlab")
R> facedata <- readMat(file.path("face_data.mat"))
R> x <- t(facedata$images)
R> y <- facedata$poses[1,]
```

We can visualize e.g., the first observation in this data set in Figure 1.

[1]The Isomap face data can be found online on https://web.archive.org/web/20160913051505/http://isomap.stanford.edu/datasets.html.

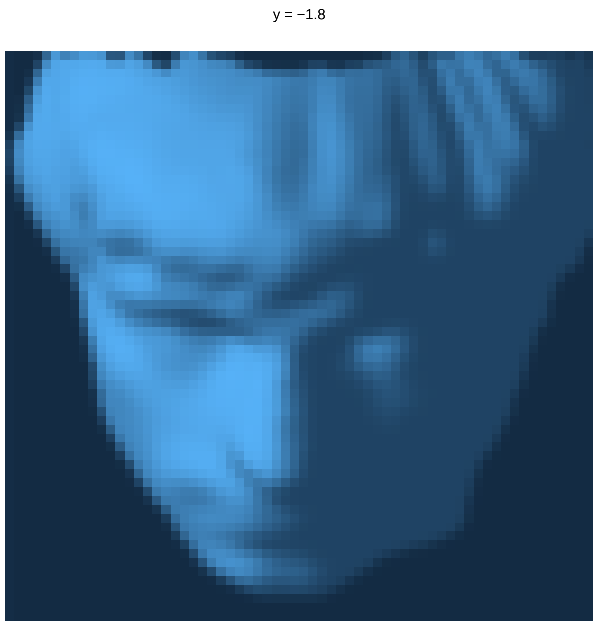

Figure 1: Image corresponding to the first observation in the *Isomap faces* data set.

This data set has the issue of many columns being almost constant, which can make estimation unstable. Given that `spar()` and `spar.cv()` ignore constant columns, we can alleviate this issue by setting all columns which have a low standard deviation to zero.

```
R> x[, apply(x, 2, sd) < 1e-2] <- 0
```

We split the data into training vs test sample

```
R> set.seed(123)
R> ntot <- length(y); ntest <- ntot * 0.25
R> testind <- sample(ntot, ntest, replace = FALSE)
R> xtrain <- as.matrix(x[-testind, ]); ytrain <- y[-testind]
R> xtest <- as.matrix(x[testind, ]); ytest <- y[testind]
```

and estimate on the training data the SPAR algorithm with cross-validation. We employ the data-driven random projection with screening based on the ridge coefficients:

```
R> library(SPAR)
R> p <- ncol(xtrain)
R> spar_faces <- spar.cv(
+  xtrain, ytrain,
+  rp = rp_cw(data = TRUE, control = list(lambda.min.ratio = 0.1)),
+  nummods = c(5, 10, 20, 50),
+  measure = "mse")
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
```

```
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
```

```
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> $lambda.min.ratio
#> [1] 0.1
#>
#> $family
#>
#> Family: gaussian
#> Link function: identity
#>
#>
#> $alpha
#> [1] 0
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
#> Warning: glmnet.fit: algorithm did not converge
R> spar_faces
#> SPAR.cv object:
#> Smallest CV-Meas 17.3 reached for nummod=50,
#>              nu=1.53e-03 leading to 3469 / 4096 active predictors.
#> Summary of those non-zero coefficients:
#>      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
#> -24.33373  -0.11138  -0.00539   0.02779   0.09874  18.26197
#>
#> Sparsest coefficient within one standard error of best CV-Meas
#>               reached for nummod=20, nu=7.45e-03
#> leading to 2430 / 4096 active
#>               predictors with CV-Meas 21.7.
#> Summary of those non-zero coefficients:
#>       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
#> -23.078412  -0.260886  -0.004426   0.040900   0.238063  16.480350
```

The plot method for 'spar.cv' objects displays by default the measure employed in the cross-validation (in this case MSE) for a grid of ν values, where the number of models is fixed to the value found to perform best in cross-validation exercise (see Figure 2). We observe that no thresholding delivers the lowest MSE.

```
R> plot(spar_faces)
```

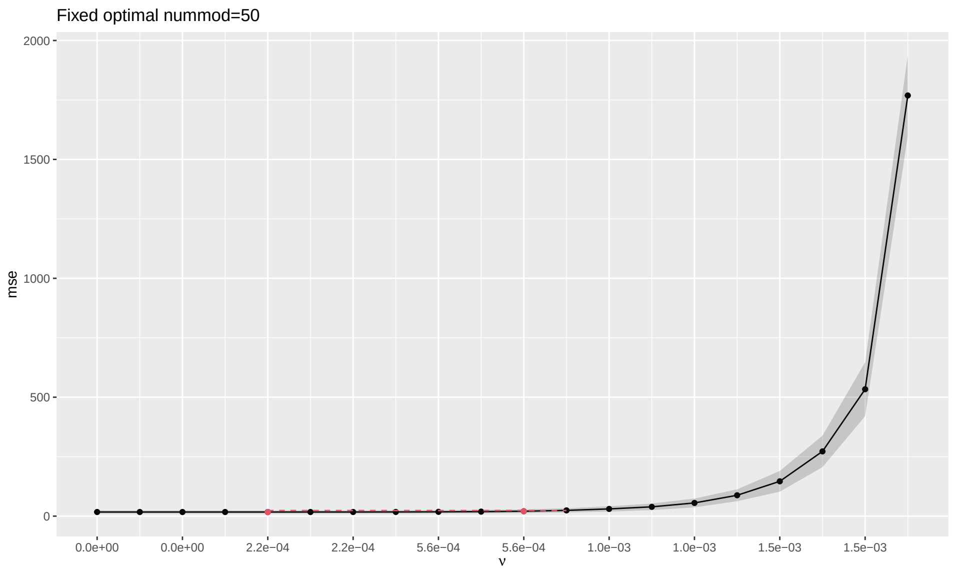

Figure 2: Plot of mean squared error over a grid of threshold values ν for fixed number of optimal models $M = 50$ for the *Isomap faces* data set. Confidence bands represent one standard deviation in the measures across the number of folds.

The coefficients of the different variables (in this example pixels) obtained by averaging over the coefficients the marginal models (for optimal λ and number of models) are given by:

```
R> face_coef <- coef(spar_faces, opt_par = "best")
R> str(face_coef)
#> List of 4
#>  $ intercept: num -22.4
#>  $ beta     : num [1:4096] 0 -0.0817 0.5942 -0.284 0.0178 ...
#>  $ nummod   : num 50
#>  $ nu       : num 0.00153
```

For a sparser solution we can compute the coefficients using `opt_par = "1se"` which leads to more sparsity and a lower number of models.

```
R> face_coef_1se <- coef(spar_faces, opt_par = "1se")
```

```
R> str(face_coef_1se)
#> List of 4
#>  $ intercept: num -22.4
#>  $ beta     : num [1:4096] 0 0.749 0.4228 -0.5957 0.0661 ...
#>  $ nummod   : num 20
#>  $ nu       : num 0.00745
```

The standardized coefficients from each of `max(nummods)` models (before averaging and before thresholding) can be plotted by setting `plot_type = "coefs"` (see Figure 3). We observe that pixels close to each other have more correlation than pixels further apart.

```
R> plot(spar_faces, plot_type = "coefs")
```

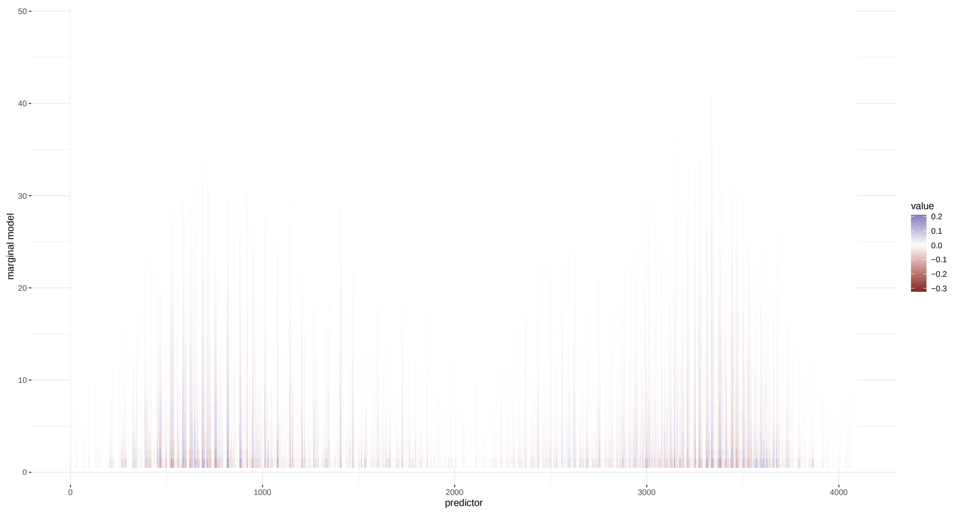

Figure 3: Coefficient plot for all variables and all $M = 50$ models in the SPAR ensemble for the *Isomap faces* data set. The coefficients are standardized, before thresholding.

The `predict()` function can be applied to the '`spar.cv`' object. We will employ the sparser solution chosen by the `opt_par = "1se"` rule:

```
R> ynew <- predict(spar_faces, xnew = xtest, coef = face_coef_1se)
```

In the high-dimensional setting it is interesting to look at the relative mean square prediction error which compares the MSE to the MSE of a model containing only an intercept:

```
R> rMSPEconst <- mean((ytest - mean(y))^2)
R> mean((ynew-ytest)^2)/rMSPEconst
#> [1] 0.0234727
```

Additionally, for this data set, one can visualize the effect of each pixel $\widehat{\beta}_j x^{\text{new}}_{i,j}$ in predicting the face orientation in a given image e.g., the 11th one in the test set. The contribution of each pixel can be visualized in Figure 4.

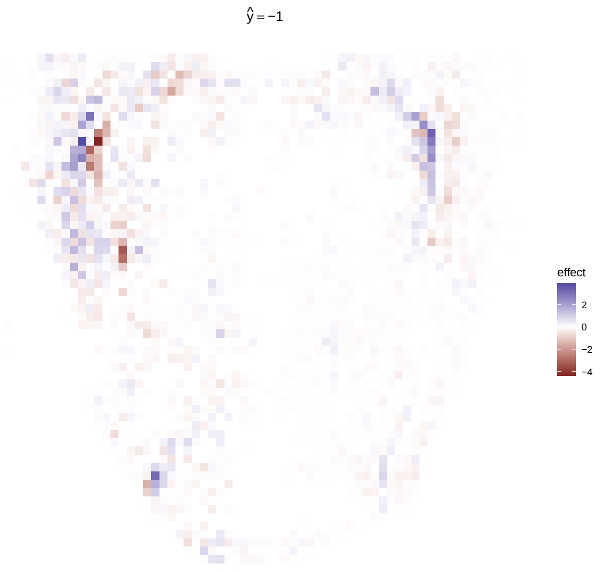

Figure 4: The effect of each pixel $\hat{\beta}_j x_{i,j}^{\text{new}}$ in predicting the face orientation in Figure ef{fig:facesplot_i1}.

5.2. Darwin data set

The Darwin dataset (Cilia, De Gregorio, De Stefano, Fontanella, Marcelli, and Parziale 2022) contains a binary response for Alzheimer's disease (AD) together with extracted features from 25 handwriting tests (18 features per task) for 89 AD patients and 85 healthy people ($n = 174$).[2]

```
R> download.file("https://archive.ics.uci.edu/static/public/732/darwin.zip",
+               "darwin.zip")
```

```
R> darwin_tmp <- read.csv(unzip("darwin.zip",  "data.csv"),
+                         stringsAsFactors = TRUE)
```

Before proceeding with the analysis, the data is screened for multivariate outliers using the DDC algorithm in package **cellWise** (Raymaekers and Rousseeuw 2023).

```
R> darwin_orig <- list(
+  x = darwin_tmp[, !(colnames(darwin_tmp) %in% c("ID", "class"))],
+  y = as.numeric(darwin_tmp$class) - 1)
R> tmp <- cellWise::DDC(
+  as.matrix(darwin_orig$x),
+  list(returnBigXimp = TRUE,
+       tolProb = 0.999,
+       silent = TRUE))
#>
#>  The final data set we will analyze has 174 rows and 446 columns.
```

[2]The data set can be downloaded from https://archive.ics.uci.edu/dataset/732/darwin

```
#>
R> darwin <- list(x = tmp$Ximp, y = darwin_orig$y)
```

The structure of the data is:

```
R> str(darwin)
#> List of 2
#>  $ x: num [1:174, 1:450] 5160 3721 2600 2130 2310 ...
#>   ..- attr(*, "dimnames")=List of 2
#>   .. ..$ : NULL
#>   .. ..$ : chr [1:450] "air_time1" "disp_index1" "gmrt_in_air1" "gmrt_on_paper1" ...
#>  $ y: num [1:174] 1 1 1 1 1 1 1 1 1 1 ...
```

We estimate the SPAR algorithm with the screening and random projection introduced in Parzer *et al.* (2024) for binomial family and logit link, using 1−area under the ROC curve as the cross-validation measure,

```
R> spar_darwin <- spar.cv(darwin$x, darwin$y,
+                        family = binomial(logit),
+                        nummods = c(5, 10, 20, 50),
+                        measure = "1-auc")
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.475814e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.32187e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
```

```
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.468526e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.394105e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.590012e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.606675e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.293884e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.513546e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.506641e-07
#>
#> $family
#>
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.462187e-07
#>
#> $family
#>
```

```
#> Family: binomial
#> Link function: logit
#>
#>
#> $alpha
#> [1] 0
#>
#> $lambda.min.ratio
#> [1] 3.509431e-07
```

We can look at the average number of active variables for a grid of ν where the number of models is fixed to the value found to perform best in cross-validation exercise by using the `plot` method for '`spar.cv`' (see Figure 5). We observe again that no thresholding achieves the best measure and translates to almost all variables being active (some variables can be inactive at $\nu = 0$ as they may never be screened). The 1-standard-error rule would however indicate that more sparsity can be introduced without too much increase in the cross-validation measure.

```
R> plot(spar_darwin, plot_type = "Val_numAct")
```

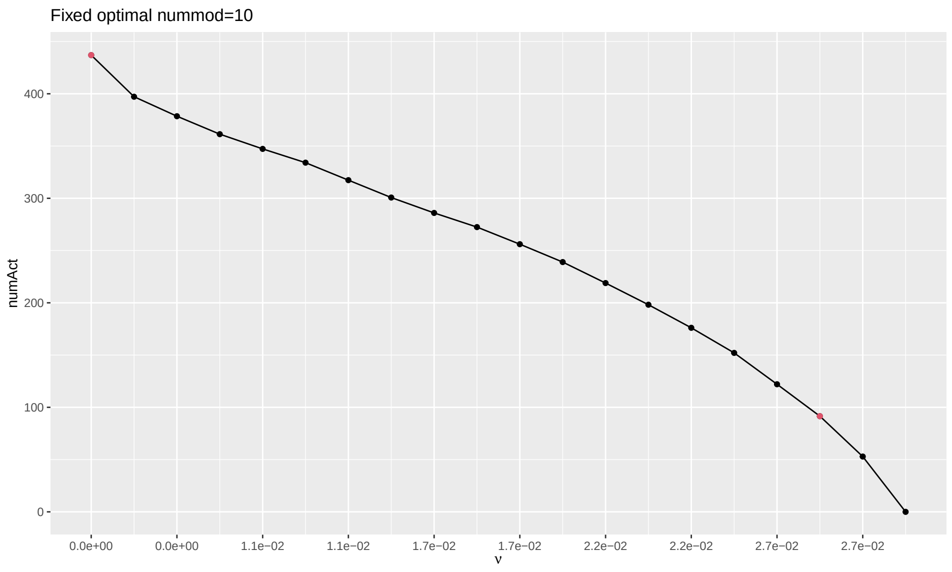

Figure 5: Average number of active variables for the grid of thresholding values ν and $M = 20$ models for the *Darwin* data set. The red points correspond to the average number of active variables for the model with the lowest cross-validation measures and to the one chosen by the 1-standard-error rule.

The predictors are ordered by task, where the first 18 covariates represent different features measured for the first task. Given that there is clear grouping in the variables in this example, we can reorder the coefficients for plotting by grouping them by feature, rather than task. We can achieve this by using reordering argument `coef_order` in method `plot` with `plot_type = "coefs"` (see Figure 6).

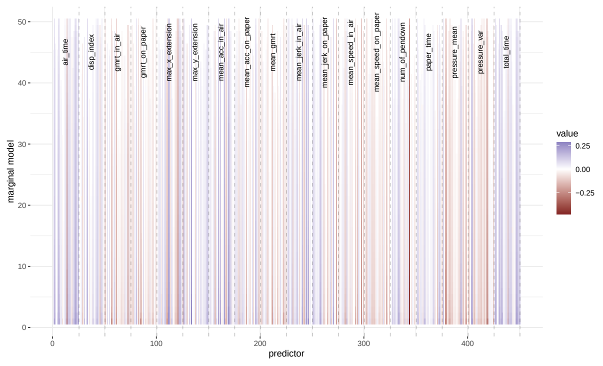

Figure 6: Coefficient plot for all variables and all $M = 50$ models in the SPAR ensemble *Darwin* data set. The coefficients are standardized, before thresholding.

In general we observe that the different features measures across different tasks have the same impact on the probability of AD (observable by the blocks of blue or red lines).

6. Conclusion

Package **SPAR** can be employed for modeling data in a high-dimensional setting, where the number of predictors is much higher than the number of observations. The package provides an implementation of the SPAR algorithm, which combines variable screening and random projection in an ensemble of GLMs. The package provides flexible classes for i) specifying the screening coefficient based on which screening should be performed (both in a classical or probabilistic fashion), ii) generating the random projection to be employed in each marginal model.

Computational details

The results in this paper were obtained using R 4.4.0.

R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/.

Acknowledgments

Roman Parzer and Laura Vana-Gür acknowledge funding from the Austrian Science Fund (FWF) for the project "High-dimensional statistical learning: New methods to advance economic and sustainability policies" (ZK 35), jointly carried out by WU Vienna University of Economics and Business, Paris Lodron University Salzburg, TU Wien, and the Austrian Institute of Economic Research (WIFO).

References

Achlioptas D (2003). "Database-Friendly Random Projections: Johnson-Lindenstrauss with Binary Coins." *Journal of Computer and System Sciences*, **66**(4), 671–687. ISSN 0022-0000. doi:10.1016/S0022-0000(03)00025-4. Special Issue on PODS 2001.

Aghila G, Siddharth R (2020). ***RandPro**: Random Projection with Classification.* doi:10.32614/CRAN.package.RandPro. R package version 0.2.2.

Ailon N, Chazelle B (2009). "The fast Johnson–Lindenstrauss transform and approximate nearest neighbors." *SIAM Journal on computing*, **39**(1), 302–322.

Bengtsson H (2022). ***R.matlab**: Read and Write MAT Files and Call MATLAB from Within R.* R package version 3.7.0, URL https://CRAN.R-project.org/package=R.matlab.

Cannings TI, Samworth RJ (2017). "Random-projection ensemble classification." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **79**(4), 959–1035. doi:10.1111/rssb.12228.

Cannings TI, Samworth RJ (2021). ***RPEnsemble**: Random Projection Ensemble Classification.* R package version 0.5, URL 10.32614/CRAN.package.RPEnsemble.

Cheng X, Wang H, Zhu L, Zhong W, Zhou H (2024). *MFSIS: Model-Free Sure Independent Screening Procedures.* R package version 0.2.1, URL https://CRAN.R-project.org/package=MFSIS.

Cho H, Fryzlewicz P (2012). "High dimensional variable selection via tilting." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **74**(3), 593–622. doi:10.1111/j.1467-9868.2011.01023.x.

Cilia ND, De Gregorio G, De Stefano C, Fontanella F, Marcelli A, Parziale A (2022). "Diagnosing Alzheimer's disease from on-line handwriting: A novel dataset and performance benchmarking." *Engineering Applications of Artificial Intelligence*, **111**, 104822. ISSN 0952-1976. doi:https://doi.org/10.1016/j.engappai.2022.104822. URL https://www.sciencedirect.com/science/article/pii/S0952197622000902.

Clarkson KL, Woodruff DP (2013). "Low Rank Approximation and Regression in Input Sparsity Time." In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, p. 81–90. Association for Computing Machinery, New York, NY, USA. ISBN 9781450320290. doi:https://doi.org/10.1145/2488608.2488620. URL https://doi.org/10.1145/2488608.2488620.

Cui H, Li R, Zhong W (2015). "Model-free feature screening for ultrahigh dimensional discriminant analysis." *Journal of the American Statistical Association*, **110**(510), 630–641.

Fan J, Feng Y, Song R (2011). "Nonparametric independence screening in sparse ultra-high-dimensional additive models." *Journal of the American Statistical Association*, **106**(494), 544–557.

Fan J, Feng Y, Wu Y (2010). "High-dimensional variable selection for Cox's proportional hazards model." In *Borrowing strength: Theory powering applications–a Festschrift for Lawrence D. Brown*, volume 6, pp. 70–87. Institute of Mathematical Statistics.

Fan J, Lv J (2007). "Sure Independence Screening for Ultra-High Dimensional Feature Space." *J Roy Stat Soc*, **B 70**.

Fan J, Samworth R, Wu Y (2009). "Ultrahigh dimensional feature selection: beyond the linear model." *The Journal of Machine Learning Research*, **10**, 2013–2038. URL https://jmlr.csail.mit.edu/papers/v10/fan09a.html.

Fan J, Song R (2010). "Sure independence screening in generalized linear models with NP-dimensionality." *The Annals of Statistics*, **38**(6), 3567 – 3604. `doi:10.1214/10-AOS798`. URL https://doi.org/10.1214/10-AOS798.

Frankl P, Maehara H (1988). "The Johnson-Lindenstrauss lemma and the sphericity of some graphs." *Journal of Combinatorial Theory, Series B*, **44**(3), 355–362. ISSN 0095-8956. `doi:https://doi.org/10.1016/0095-8956(88)90043-3`. URL https://www.sciencedirect.com/science/article/pii/0095895688900433.

Gataric M, Wang T, Samworth RJ (2019). ***SPCAvRP**: Sparse Principal Component Analysis via Random Projections (SPCAvRP).* R package version 0.4, URL 10.32614/CRAN.package.SPCAvRP.

Johnson W, Lindenstrauss J (1984). "Extensions of Lipschitz maps into a Hilbert space." *Contemporary Mathematics*, **26**, 189–206. `doi:https://doi.org/10.1090/conm/026/737400`.

Ke C (2023). "Sufficient variable screening with high-dimensional controls." *Electronic Journal of Statistics*, **17**(2), 2139 – 2179. `doi:10.1214/23-EJS2150`. URL https://doi.org/10.1214/23-EJS2150.

Li P, Hastie TJ, Church KW (2006). "Very Sparse Random Projections." In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, p. 287–296. Association for Computing Machinery, New York, NY, USA. ISBN 1595933395. `doi:https://doi.org/10.1145/1150402.1150436`. URL https://doi.org/10.1145/1150402.1150436.

Li R, Huang L, Dziak J (2022). *VariableScreening: High-Dimensional Screening for Semiparametric Longitudinal Regression.* R package version 0.2.1, URL https://CRAN.R-project.org/package=VariableScreening.

Li R, Zhong W, Zhu L (2012). "Feature screening via distance correlation learning." *Journal of the American Statistical Association*, **107**(499), 1129–1139.

Ma X, Zhang J (2016). "Robust model-free feature screening via quantile correlation." *Journal of Multivariate Analysis*, **143**, 472–480.

Mai Q, Zou H (2013). "The Kolmogorov filter for variable screening in high-dimensional binary classification." *Biometrika*, **100**(1), 229–234. URL https://doi.org/10.1093/biomet/ass062.

Mai Q, Zou H (2015). "The fused Kolmogorov filter: A nonparametric model-free screening method." *The Annals of Statistics*, **43**(4), 1471 – 1497. `doi:10.1214/14-AOS1303`. URL https://doi.org/10.1214/14-AOS1303.

Matoušek J (2008). "On variants of the Johnson–Lindenstrauss lemma." *Random Structures & Algorithms*, **33**(2), 142–156. doi:10.1002/rsa.20218.

Mukhopadhyay M, Dunson DB (2020). "Targeted Random Projection for Prediction From High-Dimensional Features." *Journal of the American Statistical Association*, **115**(532), 1998–2010. doi:https://doi.org/10.1080/01621459.2019.1677240. https://doi.org/10.1080/01621459.2019.1677240, URL https://doi.org/10.1080/01621459.2019.1677240.

Parzer R, Filzmoser P, Vana-Gür L (2024). "Data-Driven Random Projection and Screening for High-Dimensional Generalized Linear Models." *Technical Report 2410.00971*, arXiv.org E-Print Archive. doi:10.48550/arXiv.2410.00971.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, *et al.* (2011). "**scikit-learn**: Machine learning in Python." *Journal of Machine Learning Research*, **12**(Oct), 2825–2830. URL https://www.jmlr.org/papers/v12/pedregosa11a.html.

R Core Team (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Raymaekers J, Rousseeuw P (2023). *cellWise: Analyzing Data with Cellwise Outliers*. R package version 2.5.3, URL https://CRAN.R-project.org/package=cellWise.

Ryder N, Karnin Z, Liberty E (2019). "Asymmetric Random Projections." 1906.09489.

Saldana DF, Feng Y (2018). "**SIS**: An R Package for Sure Independence Screening in Ultrahigh-Dimensional Statistical Models." *Journal of Statistical Software*, **83**(2), 1–25. doi:https://doi.org/10.18637/jss.v083.i02.

Siddharth R, Aghila G (2020). "RandPro – A practical implementation of random projection-based feature extraction for high dimensional multivariate data analysis in R." *SoftwareX*, **12**, 100629. ISSN 2352-7110. doi:10.1016/j.softx.2020.100629. URL https://www.sciencedirect.com/science/article/pii/S2352711020303423.

Tay JK, Narasimhan B, Hastie T (2023). "Elastic Net Regularization Paths for All Generalized Linear Models." *Journal of Statistical Software*, **106**(1), 1–31. doi:10.18637/jss.v106.i01.

Thanei GA, Heinze C, Meinshausen N (2017). *Random Projections for Large-Scale Regression*, pp. 51–68. Springer International Publishing, Cham. ISBN 978-3-319-41573-4. doi:https://doi.org/10.1007/978-3-319-41573-4_3. URL https://doi.org/10.1007/978-3-319-41573-4_3.

Tian Y, Feng Y (2021). ***RaSEn**: Random Subspace Ensemble Classification and Variable Screening*. R package version 3.0.0, URL https://CRAN.R-project.org/package=RaSEn.

van Rossum G, *et al.* (2011). *Python Programming Language*. URL http://www.python.org.

Wang X, Leng C (2016). "High-dimensional Ordinary Least-squares Projection for Screening Variables." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **78**, 589–611. doi:https://doi.org/10.1111/rssb.12127.

Wang X, Pan W, Hu W, Tian Y, Zhang H (2015). "Conditional distance correlation." *Journal of the American Statistical Association*, **110**(512), 1726–1734.

Wanjun Liu Yuan Ke JL, Li R (2022). "Model-Free Feature Screening and FDR Control With Knockoff Features." *Journal of the American Statistical Association*, **117**(537), 428–443. doi:10.1080/01621459.2020.1783274. https://doi.org/10.1080/01621459.2020.1783274, URL https://doi.org/10.1080/01621459.2020.1783274.

Wickham H (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org.

Xu C, Chen J (2014). "The sparse MLE for ultrahigh-dimensional feature screening." *Journal of the American Statistical Association*, **109**(507), 1257–1269.

Zhu LP, Li L, Li R, Zhu LX (2011). "Model-free feature screening for ultrahigh-dimensional data." *Journal of the American Statistical Association*, **106**(496), 1464–1475.

Affiliation:

Roman Parzer
Computational Statistics (CSTAT), Institute of Statistics and Mathematical Methods in Economics
Wiedner Hauptstraße 8-10
Vienna Austria
E-mail: romanparzer1@gmail.com

Laura Vana Gür
Computational Statistics (CSTAT), Institute of Statistics and Mathematical Methods in Economics
Wiedner Hauptstraße 8-10
Vienna Austria

Peter Filzmoser
Computational Statistics (CSTAT), Institute of Statistics and Mathematical Methods in Economics
Wiedner Hauptstraße 8-10
Vienna Austria

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd