spareg: Sparse Projected Averaged Regression in R

Laura Vana-Gür 
TU Wien

Roman Parzer 
TU Wien

Peter Filzmoser 
TU Wien

Abstract

The **spareg** package for R builds ensembles of predictive generalized linear models for high-dimensional data. It employs an algorithm that combines variable screening and random projection techniques to address the computational challenges of large predictor sets. The package makes this modeling approach more accessible, as variants of the algorithm have demonstrated competitive predictive performance compared to state-of-the-art methods, while maintaining low computational costs. By implementing this modeling approach in an accessible framework, **spareg** enables users to apply methods that have shown competitive predictive performance against state-of-the-art alternatives, while at the same time keeping computational costs low.

Designed with extensibility in mind, **spareg** implements the screening and random projection techniques, as well as the generalized linear models employed in the ensemble as S3 classes with user-friendly constructor functions. This modular design allows users to seamlessly integrate and develop new procedures, making the package a versatile tool for high-dimensional applications.

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

1. Introduction

The **spareg** package for R (R Core Team 2024) offers functionality for estimating generalized linear models (GLMs) in high-dimensional settings, where the number of predictors p significantly exceeds the number of observations n i.e., $p > n$ or even $p \gg n$. To address the challenges of high dimensionality, the package implements a general algorithm which integrates variable screening methods with random projection techniques to effectively reduce the predictor space.

More specifically, **spareg** builds an ensemble of GLMs where, in each model of the ensemble, i) variables are first screened based on a measure of the utility of each predictor for the response, ii) the selected variables are then projected to a lower dimensional space (smaller than n) using a random projection matrix, iii) a GLM is estimated using the projected predictors. Finally, additional sparsity in the coefficients of the original variables can be introduced through a thresholding parameter, which together with the number of models in the ensemble can be chosen using a validation set or via cross-validation. The final coefficients are then obtained by averaging over the marginal models in the ensemble.

The motivation of such an algorithm, which performs what we call a *sparse projected averaged regression* (SPAR) for both discrete and continuous data in the GLM framework, lies in its computational efficiency. Random projection is a computationally-efficient method which

linearly maps a set of points in high dimensions into a much lower-dimensional space and random projection matrices have been traditionally generated from suitable distributions in a data-agnostic fashion. By projecting the original predictors to a dimension lower than n, estimation of the models on the reduced predictors can be done using standard methods.

However, random projection can suffer from noise accumulation for very large p, as too many irrelevant predictors are being considered for prediction purposes (Mukhopadhyay and Dunson 2020). Therefore, the screening step is advisable in order to eliminate the influence of irrelevant variables before performing the random projection, while also reducing computational costs.

The ensemble approach is motivated by the fact that, although combining variable screening with random projection effectively reduces the predictor set and computational costs, the variability introduced by random sampling can be mitigated by averaging the results from multiple iterations (Thanei, Heinze, and Meinshausen 2017).

Different variants of the algorithm have been shown to perform well in terms of prediction power on a variety of data sets. For example, Mukhopadhyay and Dunson (2020) employ the algorithm with the data-agnostic, sparse random projection of Achlioptas (2003) combined with probabilistic marginal correlation screening. Furthermore, Parzer, Filzmoser, and Vana-Gür (2024a) show that, when paired with a carefully constructed data-driven random projection, the algorithm performs superiorly in terms of predictions and variable ranking in settings with different degrees of sparsity in the coefficients and with correlated predictors.

Given the variety of screening and random projection matrices to be possibly employed in the algorithm – each with distinct advantages across different data settings – package **spareg** offers a diverse selection of screening coefficients and multiple procedures for constructing random projection matrices. Designed for flexibility, the package provides a versatile framework that allows users to extend its screening and projection techniques, as well as the GLM models employed in the ensemble with custom procedures. This is achieved through R's S3 classes and user-friendly constructor functions

The package provides methods such as `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.* 2024a), facilitates interpretability of the model output, allowing users to understand variable effects.

While, to the best of our knowledge, **spareg** offers the first implementation of the described algorithm and no other package offers the same functionality for GLMs, few other R packages focus on building ensembles where the dimensionality of the predictors is 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 an 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.

The rest of the paper is organized as follows: Section 2 provides an overview of the methods and the methodological details of the implemented algorithm. The package is described in Section 3 and Section 4 exemplifies how a new screening coefficient, a new random projection

and a new marginal model can be integrated in the package. Section 5 concludes.

2. Methods

Throughout the section we assume to observe high-dimensional data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ is a predictor vector and $y_i \in \mathbb{R}$ is the response, with $p \gg n$. The predictor vectors are collected in the rows of the predictor matrix $X \in \mathbb{R}^{n \times p}$.

2.1. Variable screening

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).

The field of variable screening is an active area of research, with numerous methods developed to address different data settings. While a comprehensive review is beyond the scope of this paper, this section provides a concise and selective overview of key approaches for variable screening in high-dimensional settings. A classic approach is sure independence screening (SIS), proposed by Fan and Lv (2007), which uses the vector of marginal empirical correlations $\hat{\boldsymbol{\omega}} = (\hat{\omega}_1, \ldots, \hat{\omega}_p)^\top \in \mathbb{R}^p, \hat{\omega}_j = \text{Cor}(X_j, y)$, where $\boldsymbol{y}$ is the $(n \times 1)$ vector of responses and X_j is the j-th column of the matrix of predictors, to screen predictors in a linear regression setting by selecting the variable set $\mathcal{A}_\delta$ which contains all variables for which $|\hat{\omega}_j| > \delta$, where δ is a suitable threshold. Under certain technical conditions, this screening coefficient has the *sure screening property*, i.e., that the set of truly active variables is included in $\mathcal{A}_{\delta_n}$ with probability converging to one as $n \to \infty$. Extensions to SIS include modifications for GLMs (Fan and Song 2010), where screening is performed based on the log-likelihood $\ell(.)$ or the slope coefficient of the GLM containing only X_j as a predictor: $\hat{\omega}_j =: \text{argmin}_{\beta_j \in \mathbb{R}} \text{min}_{\beta_0 \in \mathbb{R}} \sum_{i=1}^n -\ell(\beta_j, \beta_0; y_i, x_{ij})$, where x_{ij} is the j-th entry of the vector x_i.

However, both mentioned 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, Cho and Fryzlewicz (2012) propose using the tilted correlation, i.e., the correlation of a tilted version of X_j with $\boldsymbol{y}$ where the effect of other variables is reduced. Wang and Leng (2016) propose to take into account the correlation among the predictors by employing the high-dimensional ordinary least squares projection (HOLP), which is a ridge estimator where the penalty term converges to zero. 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 $\mathsf{P}(X_j \leq x|y)$ (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.

The R landscape for variable screening techniques is very rich. 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 independent and identically distributed (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 Yuan Ke and Li 2022), among others. Additional packages implement specific procedures but their review is beyond the scope of the current paper.

Package **spareg** allows the integration of such (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 δ, predictors are sampled with probabilities proportional to their screening coefficient (see Mukhopadhyay and Dunson 2020; Parzer *et al.* 2024a). Note that this is different than the random subspace sampling employed in, e.g., random forests.

2.2. Random projection tools

Package **spareg** 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 different random projection techniques and an overview of existing software implementing random projections.

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

The goal is to choose a random map Φ that satisfies the JL lemma with high probability

given that it fulfills certain technical conditions. 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 where $\Phi_{ij} \overset{iid}{\sim} N(0, 1/\sqrt{\psi})$ 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 the Haar measure to generate random orthogonal matrices (Cannings and Samworth 2017) or a non-sub-Gaussian distribution like the standard Cauchy, proposed by Li *et al.* (2006) 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" data-agnostic random projection or to directly inform the random projection procedure. Cannings and Samworth (2017) rely on the former approach and 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.* (2024a) 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. Parzer, Filzmoser, and Vana-Gür (2024b) also provide in the linear regression a theoretical bound on the expected gain in prediction error in using a projection which incorporates information about the true regression coefficients compared to a conventional random projection. Motivated by this result, they propose to construct a projection matrix using the sparse embedding matrix of Clarkson and Woodruff (2013), where the random diagonal elements are replaced in practice by a ridge coefficient with a minimal λ penalty. This method has the advantage of approximately capturing the true regression coefficients in the span of the random projection, i.e., it ensures that the true regression coefficients can be recovered approximately after the projection. 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.

Several packages in R provide functionality for random projections. For instance, package **RandPro** (Aghila and Siddharth 2020; Siddharth and Aghila 2020) allows a Gaussian random matrix (i.e, where entries are simulated from a standard normal), a sparse matrix (Achlioptas 2003; Li *et al.* 2006) or a matrix generated using the equal probability distribution with the elements $\{-1, 1\}$, to be applied to the predictor matrix before employing one of k nearest neighbors, support vector machine or naive Bayes classifier on 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 **RPEnsem-**

bleR (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 matrices, namely Gaussian random matrix and sparse random matrix.

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 $\boldsymbol{z}_i = \Phi \boldsymbol{x}_i \in \mathbb{R}^m$ together with the response arise from a GLM with the response having conditional density from a (reproductive) exponential dispersion family of the form

$$f(y_i|\theta_i, \phi) = \exp\Big\{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)\Big\}, \quad g(\mathsf{E}[y_i|\boldsymbol{z}_i]) = \gamma_0 + (\Phi\boldsymbol{x}_i)^\top\boldsymbol{\gamma} =: \eta_i,$$

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 $\boldsymbol{z}_i$ depends on a linear combination of the predictors through a (invertible) link function $g(.)$, where $\gamma_0 \in \mathbb{R}$ is the intercept and $\boldsymbol{\gamma} \in \mathbb{R}^m$ is a vector of regression coefficients for the m projected predictors. We can see that the original coefficients of the predictors $\boldsymbol{x}_i$ can be obtained as: $\boldsymbol{\beta} = \Phi^\top\boldsymbol{\gamma}$.

Estimates for $\gamma_0 \in \mathbb{R}$ and $\boldsymbol{\gamma} \in \mathbb{R}^m$ can be obtained by maximum likelihood, as generally one chooses $m < n$. However, even if $m < n$ it might still be desirable to add a (e.g., small L_2) penalty to the likelihood of the marginal models – for the binomial family this can alleviate problems related to separation. Moreover, if outlier influence should be reduced, M- or trimmed estimators could be employed for obtaining robust estimates of the regression coefficients (see e.g., Cantoni and Ronchetti 2001).

2.4. SPAR algorithm

We present the general algorithm for sparse projected averaged regression (SPAR) implemented in package **spareg**.

1. Choose family with corresponding log-likelihood $\ell(.)$ and link.
2. Standardize the $(n \times p)$ matrix of predictors X for all families and the vector of responses $\boldsymbol{y}$ for the Gaussian family by subtracting the sample mean and dividing by the sample standard deviation of each variable.
3. Calculate screening coefficients $\hat{\boldsymbol{\omega}}$.
4. For $k = 1, \ldots, M$ models:

(a) If $p > p_0$, where p_0 is the number of variables to be screened, screen p_0 predictors based on the screening coefficient $\hat{\boldsymbol{\omega}}$, which yields for model k the screening index set $I_k = \{j_1^k, \ldots, j_{p_0}^k\} \subset \{1, \ldots, p\}$; 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 p_0 variables with the highest $|\hat{\omega}_j|$. If $p < p_0$, perform no screening and $I_k = \{1, \ldots, p\}$.

(b) Project the screened variables to a random dimension $m_k \sim \text{Unif}\{m_l, \ldots, m_u\}$ using *projection matrix* Φ_k to obtain $Z_k = X_{\cdot I_k} \Phi_k^\top \in \mathbb{R}^{n \times m_k}$, where $X_{\cdot I_k}$ contains the columns in X having a column index in I_k.

(c) Fit a *GLM*-type model of $\boldsymbol{y}$ on Z_k to obtain estimated coefficients $\hat{\boldsymbol{\gamma}}^k \in \mathbb{R}^{m_k}$ and $\hat{\boldsymbol{\beta}}^k_{I_k} = \Phi_k^\top \hat{\boldsymbol{\gamma}}^k$, $\hat{\boldsymbol{\beta}}^k_{\bar{I}_k} = 0$.

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 and employing a loss function $K(M, \nu)$ on the test set

$$(M_{\text{best}}, \nu_{\text{best}}) = \text{argmin}_{M,\nu} K(M, \nu).$$

7. Combine models of the ensembles either via the coefficients by using a *simple average* of the regression coefficients $\hat{\boldsymbol{\beta}} = \sum_{k=1}^M \hat{\boldsymbol{\beta}}^k / M$ (this results in averaging of the models on the link level) or via the fitted values by taking a *simple average* of $\hat{y} = \sum_{k=1}^M g^{-1}(\boldsymbol{x}_i^\top \hat{\boldsymbol{\beta}}^k)/M$.

8. Output the averaged estimated coefficients and predictions for the chosen M and ν.

In order to choose default values for p_0, m_l, m_u as well as the maximal number of considered models M, we rely on the analysis in Parzer *et al.* (2024b). They find that, when using their proposed data-driven random projection and HOLP screening coefficient, the gain in predictive performance when using more than $M = 20$ models is marginal. We use therefore $M = 20$ as a default value in the algorithm. Regarding the number of screened variables, they propose choosing $p_0 = c \cdot n$ as a multiple of n (thus independent of p) and find that the best results are achieved for $2 \leq c \leq 4$. We thus set in the algorithm $p_0 = 2n$ as a default value. Finally, Mukhopadhyay and Dunson (2020) propose using $m_l = 2\log(p)$ and $m_u = 3n/4$ while Parzer *et al.* (2024b) use slightly smaller goal dimensions ($m_l = \log(p)$, $m_u = n/2$), to reduce the dimension of the marginal models to be estimated. On the other hand, for random projection matrices satisfying the JL lemma, m_l can be derived from the theoretical bounds for preserving the distances between all pairs of points approximately. We employ as default values the ones proposed in Parzer *et al.* (2024b).

A further modification of the algorithm above is to employ part of the data for estimating the screening coefficients (Step 3) and the remaining data to estimate the ensemble models (Step 4). Such a strategy could avoid issues related to overfitting. Even though Parzer *et al.* (2024b) find that in the linear regression case such a splitting approach does not improve performance, the implementation in **spareg** allows for data splitting.

3. Software

Package **spareg** (Vana-Gür, Parzer, and Filzmoser 2025) is available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=spareg.

```
R> install.packages("spareg")
```

and loaded by

```
R> library("spareg")
```

In this section we rely for illustration purposes on a simulated example data set which contains $n = 200$ observations of a continuous response `y` and $p = 2000$ predictors `x` which can be used as a training data set and $n = 100$ observations to be used as a test set.

```
R> set.seed(1234)
R> example_data <- simulate_spareg_data(n = 200, p = 2000, ntest = 100)
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
```

The function `simulate_spareg_data()` simulates data from a linear regression model. Using the default values, data is generated using $\sigma^2 = 83$, an intercept $\mu = 1$ and β coefficients with 100 non-zero entries, where the non-zero entries are uniformly sampled from $\{-3, -2, -1, 1, 2, 3\}$.

3.1. Main functions and their arguments

The two main functions for fitting the SPAR algorithm are:

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

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

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

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 where $n < p$,
- `y` numeric response vector of length n,
- `family` object from `stats::family()`; defaults to `gaussian()`;
- `model` an object of class '`sparmodel`' which specifies the model employed for each element of the ensemble. Defaults to `spar_glm()` for Gaussian family with identity link and to `spar_glmnet()` for all other family-link combinations. The latter uses the **glmnet** package to penalize the marginal models with a small ridge penalty in order to increase stability in the coefficients.
- `rp` an object of class '`randomprojection`'. Defaults to `NULL`, in which case `rp_cw(data = TRUE)` is used.
- `screencoef` an object of class '`screencoef`'. Defaults to `NULL`, in which case no screening is employed.
- `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` specifies the measure $K(\nu, M)$ 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()`). The default value for `measure` is `"deviance"`, which is available for all families. Other options are mean squared error `"mse"` or mean absolute error `"mae"` (between responses and predicted conditional means, for all families), `"class"` (misclassification error) and `"1-auc"` (one minus area under the ROC curve) both just for binomial family.
- `parallel` assuming a parallel backend is loaded and available, a logical indicating whether the function should use it for parallelizing the estimation of the marginal models. Defaults to `FALSE`.

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 I_k 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`.
- `RPM` 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 a lightweight version of `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. If data-driven random projections are employed, only the data to be used in the random projection will be updated in each fold iteration with the corresponding training data. More details will be provided 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,

```
screen_*(..., control = list())
```

which take as arguments `...` (to be saved as attributes of the object) and `control` (a list of controls to be used in the main function for computing the screening coefficients).

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

- `screen_marglik()` – computes the screening coefficients by the coefficient of X_j for $j = 1, \ldots, p$ 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`, `offset` – e.g., `screen_marglik(control = list(family = binomial(probit)))`). Note that if `family` is not provided in `control`, the `family` used in `spar()` will be used.

- `screen_cor()` – computes the screening coefficients by the correlation between $\boldsymbol{y}$ and X_j using the function `stats::cor()`. It allows to pass a list of controls through the `control` argument to `stats::cor()` – e.g., `screen_cor(control = list(method = "spearman"))`.

- `screen_glmnet()` – computes by default the ridge coefficient where the penalty λ is very small (see Parzer *et al.* 2024a, for clarification).

$$\hat{\omega} =: \text{argmin}_{\beta \in \mathbb{R}^p} \text{min}_{\beta_0 \in \mathbb{R}} \sum_{i=1}^{n} -\ell(\beta; y_i, \boldsymbol{x}_i) + \frac{\varepsilon}{2} \sum_{j=1}^{p} \beta_j^2, \; \varepsilon > 0$$

  The function relies on `glmnet::glmnet()`. It uses by default $\alpha = 0$ and a small `lambda.min.ratio`. The optimal penalty is chosen using the recommendations in Parzer

et al. (2024a), namely choosing the smallest penalty for which the deviance ratio (the fraction of null deviance explained) is less than 0.99 for the Gaussian family and 0.8 for other families. It, however, allows to pass a list of controls through the `control` argument to `glmnet::glmnet()` – e.g., `screen_glmnet(control = list(alpha = 0.5))`.

As mentioned above, arguments related to the screening procedure can be passed to the `screen_*()` function 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; if not specified, 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; if not specified, the whole data will be used for estimating the screening coefficient and the marginal 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"`.
- `reuse_in_rp` logical – indicates whether the screening coefficient should be reused at a later stage in the construction of the random projection. Defaults to `FALSE`.

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 the following arguments: `x` – the matrix of standardized predictors – and `y` – the vector of (standardized in the Gaussian case) responses, and the argument `object`, which is a '`screencoef`' object itself. 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`' object 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"

$generate_fun
function (y, x, object)
{
    control <- object$control
    if (is.null(control$family)) {
        control$family <- eval(parse(text = attr(object, "family_string")))
    }
    coefs <- apply(x, 2, function(xj) {
        glm_res <- do.call(function(...) glm(y ~ xj, ...), control)
        glm_res$coefficients[2]
    })
    coefs
}
<environment: namespace:spareg>

$control
list()

attr(,"type")
[1] "prob"
attr(,"reuse_in_rp")
[1] FALSE
```

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 imple-

mented as S3 classes '`randomprojection`' and are created by functions which take `...` and a list of controls `control` as arguments:

```
rp_*(..., control = list())
```

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

- `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). The value of ψ can be passed through the `control` argument and defaults to `psi = 1` e.g., `rp_sparse(control = list(psi = 1/3))`.
- `rp_cw()` – sparse embedding random projection in Clarkson and Woodruff (2013). This matrix is constructed as $\Phi = BD \in \mathbb{R}^{m\times p}$, where B is a $(p \times p)$ binary matrix, where for each column j an index is uniformly sampled from $\{1, \dots, m\}$ and the corresponding entry is set to one, and D is a $(p \times p)$ diagonal matrix, with entries $d_j \sim \text{Unif}(\{-1, 1\})$. If specified as `rp_cw(data = TRUE)`, the random elements on the diagonal are replaced by the ridge coefficients with a small penalty, as introduced in Parzer *et al.* (2024a).

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 and are present in all '`randomprojection`' objects:

- `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 $n/2$.

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, Achlioptas (2003) show $m_0 = \log n(4 + 2\tau)/(\epsilon^2/2 - \epsilon^3/3)$ for probability $1 - n^{-\tau}$.

The `rp_*()` functions return an object of class '`randomprojection`' which is a list of five elements. The most important three elements are:

- a character `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;
  - 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.

  - `x` the vector of standardized predictors–can be `NULL` if the random projection to be generated is data-agnostic;
  - `y` the vector of (standardized) responses–can be `NULL` if the random projection to be generated is data-agnostic.

  It returns a matrix or a sparse matrix of class '`dgCMatrix`' of the **Matrix** package (Bates, Maechler, and Jagan 2024) with `m` rows and `length(included_vector)` columns.

- `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 the case where the random projection should incorporate some information related to the data, two more elements can be potentially relevant.

- Function `update_fun()` updates the '`randomprojection`' object with relevant information from the arguments of the `spar()` function call. If it is not provided, it defaults to updating the '`randomprojection`' object with a further attribute `family_string` which is character version of the `family` argument (e.g., in the default case it will be `"gaussian(identity)"`). In certain constructions, such as the one in Parzer *et al.* (2024a), certain data dependent quantities need to be computed only once, not every time a random projection is generated. For example, in Parzer *et al.* (2024a), the ridge coefficients are estimated once at the beginning of the algorithm and reused in each projection. In such cases, function `update_fun()` can be employed to add data information as attributes of the '`randomprojection`' object to be subsequently used in the `generate_fun()` function. If specified by the user, this function should take only `...` as an argument. Internally in the `spar()` function, all arguments of `spar()` are passed to this function. It should return a '`randomprojection`' object.

- In the case where a list of predefined `RPMs` is provided in `spar()`, for the data driven random projections we allow the user to specify whether some parts of the given `RPMs` should be updated with the provided data. This is possible through another optional function `update_rpm_w_data()`. This is particularly relevant for the cross-validation procedure, which employs the random projection matrices generated by calling the `spar()` function on the whole data set before starting the cross-validation exercise. For example, in our implementation of the data-driven `rp_cw(data = TRUE)`, in each fold, we only update the list of `RPMs` by adjusting the diagonal elements to the vector of screening coefficients computed on the training data for the current 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.

For illustration purposes, consider the implemented function `rp_gaussian()`, which generates 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:
```

```
* 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"

$generate_fun
function (rp, m, included_vector, x = NULL, y = NULL)
{
    p <- length(included_vector)
    control_rnorm <- c(rp$control[names(rp$control) %in% names(formals(rnorm))],
        attributes(rp)[names(attributes(rp)) %in% names(formals(rnorm))])
    control_rnorm <- control_rnorm[!duplicated(names(control_rnorm))]
    vals <- do.call(function(...) rnorm(m * p, ...), control_rnorm)
    RM <- matrix(vals, nrow = m, ncol = p)
    return(RM)
}
<environment: namespace:spareg>

$update_fun
function (...)
{
    args <- list2(...)
    if (is.null(attr(args$rp, "family"))) {
        family_string <- paste0(args$family$family, "(", args$family$link,
            ")")
        attr(args$rp, "family_string") <- family_string
    }
    args$rp
}
<environment: namespace:spareg>

$update_rpm_w_data
NULL

$control
list()
```

The `generate_fun()` function 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, the random projection does not use any data information and we are only interested in the length of this vector.

The function `update_fun()` only converts the '`family`' object to a string and adds it as an attribute. Function `update_rpm_w_data()` is `NULL` as this random projection is data-agnostic.

3.4. Marginal models

The package provides a class '`sparmodel`' for the marginal model to be fitted for each element of the ensemble. The framework assumes that model employs a linear predictor, i.e., a linear combination of the projected variables.

Similar to the objects for random projection and screening coefficients, the functions which create these objects have arguments `...` (to be saved as attributes) and `control` (to be used in the main function for building the model).

The two functions implemented are `spar_glmnet()`, which allows regularized GLMs as marginal models using function `glmnet::glmnet()` (where the default is to estimate a ridge regression with the small penalty value), and `spar_glm()` which estimates unregularized GLMs using `stats::glm()`.

An object of class '`sparmodel`' is a list with elements:

- a character `name`,
- `model_fun()` – a function which takes `y` (the vector of standardized responses), `z` (the matrix of reduced predictors) and a further argument which is the object of class '`sparmodel`' itself. It returns a list of two elements `gammas` which contains the vector of coefficients and the value of the `intercept`.
- `update_fun()` – an optional function which can add further attributes to the '`sparmodel`' object which is called at the beginning of the SPAR algorithm. This function returns the '`sparmodel`' object after modifying it. In the case of function `spar_glmnet()` this function manipulates the '`family`' object in a way which is convenient for function `glmnet::glmnet()`.[1]

The default is to use `spar_glm()` for Gaussian family with identity link and `spar_glmnet()` for the other families.

3.5. 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 summary of the non-zero coefficients. We estimate the SPAR algorithm with no screening and `rp_cw(data = TRUE)` (default).

```
R> set.seed(12)
R> spar_res <- spar(example_data$x, example_data$y,
```

[1] In the case of families Gaussian, binomial and Poisson with canonical link, the family object is replaced by a string containing the name of the family. This leads to **glmnet** using the faster specialized algorithms rather than the general algorithm implemented for all '`family`' objects.

```
+    xval = example_data$xtest, yval = example_data$ytest,
+    nummods = c(5, 10, 15, 20, 25, 30))
R> spar_res

spar object:
Smallest Validation Measure of 3.00e+04 reached for nummod=5,
             nu=4.05e-03 leading to 1017 / 2000 active predictors.
Summary of those non-zero coefficients:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.56404 -0.18048  0.10464  0.02842  0.20629  0.60673
```

For 'spar.cv' it also provides the same information for the (ν, M) combination chosen by the one-standard error rule.

```
R> spar_cv <- spar.cv(example_data$x, example_data$y,
+    nummods = c(5, 10, 15, 20, 25, 30))
R> spar_cv

spar.cv object:
Smallest CV-Meas 5042.9 reached for nummod=5, nu=0.00e+00 leading
  to 2000 / 2000 active predictors.
Summary of those non-zero coefficients (non-standardized):
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.56802 -0.09134  0.02111  0.01693  0.12608  0.60687

Sparsest coefficient within one standard error of best CV-Meas reached for
  nummod=5, nu=4.92e-03
leading to 858 / 2000 active predictors with CV-Meas 6556.6.
Summary of those non-zero coefficients (non-standardized):
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.56802 -0.18845  0.08795  0.02941  0.22233  0.60687
```

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 2.94
 $ beta     : num [1:2000] 0 0 0.462 0 0 ...
 $ nummod   : num 5
 $ nu       : num 0.00405
```

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 the sparsest vector of coefficients within one standard deviation of that optimal cross-validation `measure`. This argument is ignored when `nummod` and `nu` are given.

```
R> str(coef(spar_cv))

List of 4
 $ intercept: num 2.47
 $ beta     : num [1:2000] 0.0639 -0.0615 0.4608 0.0435 0.024 ...
 $ nummod   : num 5
 $ nu       : num 0
```

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 $y - \widehat{y}$, where $\widehat{y}$ 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 using a color scale.

- `plot_along` – one of `c("nu","nummod")`; for `plot_type = "Val_Measure"` as well as for `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. This argument must be provided for the plot of residuals and fitted values, 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. This argument must be provided for the plot of residuals and fitted values, 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 `seq_len(p)`.

The four plots for the '`spar`' object are produced with the code below and shown in Figure 1.

```
R> plot(spar_res)
R> plot(spar_res, plot_type = "Val_numAct")
R> plot(spar_res,  plot_type = "coefs")
R> plot(spar_res, plot_type = "res-vs-fitted", xfit = example_data$xtest,
+    yfit = example_data$ytest)
```

For class '`spar.cv`' there is the extra argument `opt_par = c("best", "1se")` which is only used for `plot_type = "res-vs-fitted"` and indicates whether the predictions should be based on coefficients using the best (ν, M) combination or using the combination 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).

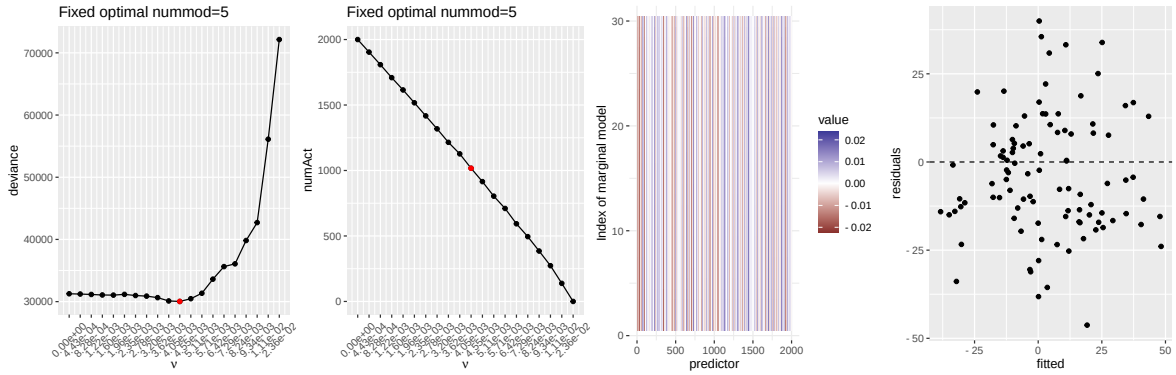

Figure 1: `plot` methods for '`spar`' object. The red dots in the first two figures show the best ν chosen on the validation set for the optimal number of models $M = 5$.

3.6. Parallelization

The package supports parallelization in the estimation of the marginal models in the ensemble via package **foreach** (Microsoft and Weston 2022). This is possible by setting the parameter `parallel = TRUE` in `spar()` or `spar.cv()`, assuming that a parallel backend for **foreach** is registered using the `registerDoParallel()` function of package **doParallel** (Microsoft Corporation and Weston 2022). In general, the parallelization seems to pay off especially for data sets with a larger number of observations n. A minimal example with a correlation-based probabilistic screening and a Gaussian random projection matrix is provided below:

```
R> set.seed(1234)
R> example_data3 <- simulate_spareg_data(n = 1000, p = 2000, ntest = 1000)
R> library(doParallel)
R> cl <- makeCluster(2, type = "PSOCK")
R> registerDoParallel(cl)
R> spar_res_par <- spar(example_data3$x, example_data3$y,
+    screencoef = screen_cor(), rp = rp_gaussian(),
+    nummods = 50, parallel = TRUE)
R> stopCluster(cl)
```

4. Extensibility

4.1. Screening coefficients

We exemplify how screening coefficients implemented in package **VariableScreening** can easily be incorporated in the framework of **spareg**.

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

```
R> generate_scr_sirs <- function(y, x, object) {
+    res_screen <- do.call(function(...)
```

```
+      VariableScreening::screenIID(x, 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 screening 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 the helper `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. The employed random projection is `rp_sparse()` where we set $\psi = 1/\sqrt{p}$, as proposed by Li *et al.* (2006).

```
R> set.seed(123)
R> spar_example <- spar(example_data$x, example_data$y,
+    screencoef = screen_sirs(type = "fixed",
+      control = list(method = "SIRS")),
+    rp = rp_sparse(psi = 1/sqrt(ncol(example_data$x))), measure = "mse")
R> spar_example

spar object:
Smallest Validation Measure of 1.48e+02 reached for nummod=20,
             nu=0.00e+00 leading to 400 / 2000 active predictors.
Summary of those non-zero coefficients:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.68568 -0.08569  0.07681  0.08965  0.24606  0.96782
```

4.2. Random projections

We exemplify how new random projections can be implemented in the framework of **spareg**. 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 take the projection matrix to be the transpose of the matrix of left singular vectors in the singular value decomposition of Q. The helper function below simulates matrices of size $m \times p$ 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), complete = FALSE)
+    t(RM)
+  }
```

Moreover, Cannings and Samworth (2017) 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. For each model k, B_2 Haar 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. (Note that, while the B_2 random projections are data-agnostic, the whole data is needed in finding the best random projection.) We will generate the random projections in the following way. For each model k, we use a proportion ξ of the data as a test set and for $b = \{1, \ldots, B_2\}$ we generate a Haar random projection. We then estimate a ridge regression on the training data and compute the misclassification error for the binomial family and MSE for all other families on the test set. Finally, the best out of B_2 projections in terms of minimizing the loss on the test set is chosen.

We can start implementing such a random projection in **spareg** by the following steps. First, there are no data quantities which are to be used in all possible random projections so we do not need to specify a function `update_fun()`. However, we can use `update_fun()` to update the '`randomprojection`' object with further information related to the family. Given that we will rely again on `glmnet::glmnet()` in each iteration b, the family object is modified to a string containing the name of the family for families Gaussian, binomial and Poisson with canonical link (this will lead to using faster specialized algorithms). This modified family is added to the list of controls as `fit_family`. We also set by default $B_2 = 50$, $\xi = 0.25$ and $\alpha = 0$ (in the **glmnet** model to be estimated inside the function generating the random projection).

```
R> update_rp_cannings <- function(...) {
+    args <- rlang::list2(...)
+    if (is.null(args$rp$control$family)) {
+      family_string <- paste0(args$family$family,
+                              "(", args$family$link, ")")
+      args$rp$control$family_string <- family_string
+      family <- args$family
+      fit_family <- switch(family$family,
+        "gaussian" = if (family$link == "identity") "gaussian" else family,
+        "binomial" = if (family$link == "logit") "binomial" else family,
+        "poisson"  = if (family$link == "log") "poisson" else family,
+        family)
+      args$rp$control$fit_family <- fit_family
+    }
+    if (is.null(args$rp$control$alpha)) args$rp$control$alpha <- 1
+    if (is.null(args$rp$control$B2))  args$rp$control$B2 <- 50
+    if (is.null(args$rp$control$xi)) {
+      args$rp$control$xi <- 0.25
+    } else {
+      stopifnot("xi must be between 0 and 1."=
+        args$rp$control$xi >= 0 &  args$rp$control$xi<= 1)
```

```
+    }
+    args$rp
+  }
```

As we can see, values for `xi` and `B2` can be passed by the user through the `control` argument, and if they are not specified, they are set to the default values.

Next we specify the function for generating the random projection.

```
R> generate_cannings <- function(rp, m, included_vector, x, y) {
+    xs <- x[, included_vector]
+    n <- nrow(x);  p <- ncol(xs)
+    B2 <- rp$control$B2;  xi <- rp$control$xi
+    id_test <- sample(n, size = n * xi)
+    xtrain <- xs[-id_test, ];  xtest <- xs[id_test,]
+    ytrain <- y[-id_test];  ytest <- y[id_test]
+    control_glmnet <-
+      rp$control[names(rp$control) %in% names(formals(glmnet::glmnet))]
+    best_val <- 1e5
+    family <- eval(parse(text = rp$control$family_string))
+    for (b in seq_len(B2)) {
+      RM <- simulate_haar(m, p)
+      xrp <- tcrossprod(xtrain, RM)
+      mod <- do.call(function(...)
+        glmnet::glmnet(x = xrp, y = ytrain,
+          family = rp$control$fit_family, ...),
+        control_glmnet)
+      coefs <- coef(mod, s = min(mod$lambda))
+      eta_test <- (cbind(1, tcrossprod(xtest, RM)) %*% coefs)
+      pred <- family$linkinv(as.vector(eta_test))
+      out_perf <-  ifelse(family$family == "binomial",
+        mean(((pred > 0.5) + 0) != ytest),
+        mean((pred - ytest)^2))
+      if (out_perf < best_val) {
+        best_val <- out_perf; best_RM <- RM
+      }
+      rm(RM)
+    }
+    return(best_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_fun = update_rp_cannings
+  )
```

We can now estimate SPAR for a binomial model, where we transform the response to a binary variable. We simulate again data from a linear model and then transform the response to a binary scale:

```
R> set.seed(1234)
R> example_data2 <- simulate_spareg_data(n = 100, p = 1000, ntest = 100)
R> ystar <- (example_data2$y > 0) + 0
R> ystarval <- (example_data2$ytest > 0) + 0
```

We use 50 models (which is in line to recommendations in Cannings and Samworth 2017), and no screening procedure. Moreover, we do not perform any thresholding:

```
R> set.seed(12345)
R> spar_example_1 <- spar(x = example_data2$x, y = ystar,
+    family = binomial(),
+    rp = rp_cannings(control = list(lambda.min.ratio = 0.01)),
+    nus = 0, nummods = 50,
+    xval = example_data2$xtest, yval = ystarval,
+    measure = "class")
```

Using the data-driven `rp_cw()`:

```
R> set.seed(12345)
R> spar_example_2 <- spar(x = example_data2$x, y = ystar,
+    family = binomial(), rp = rp_cw(data = TRUE),
+    nus = 0, nummods = 50, xval = example_data2$xtest, yval = ystarval,
+    measure = "class")
```

We can now compare the two approaches by looking at the minimum measure `Meas` achieved on the validation set:

```
R> spar_example_1$val_res[which.min(spar_example_1$val_res$Meas), ]

  nnu nu nummod numAct Meas
1   1  0     50   1000  0.2

R> spar_example_2$val_res[which.min(spar_example_2$val_res$Meas), ]

  nnu nu nummod numAct Meas
1   1  0     50   1000 0.19
```

4.3. Marginal models

Finally, we illustrate how to implement a new marginal model in **spareg**. We consider estimating a robust GLM as a marginal model using the package **robustbase** (Todorov and Filzmoser 2009).

We start by defining the `model_fun()` function, where `y` is the vector of responses, `z` is the matrix of reduced predictors and `object` is a '`sparmodel`' object. In the case of a linear model, we rely on the function `robustbase::lmrob()`, for other family-links we use `robustbase::glmrob()`.

```
R> model_glmrob <- function(y, z, object) {
+    requireNamespace("robustbase")
+    fam <- object$control$family
+    if (fam$family == "gaussian" & fam$link == "identity") {
+      glmrob_res <- do.call(function(...)
+        robustbase::lmrob(y ~ as.matrix(z), ...),
+        object$control)
+    } else {
+      glmrob_res <- do.call(function(...)
+        robustbase::glmrob(y ~ as.matrix(z), ...),
+        object$control)
+    }
+    intercept <- coef(glmrob_res)[1]
+    gammas <- coef(glmrob_res)[-1]
+    list(gammas = gammas, intercept = intercept)
+  }
```

We then construct a `spar_glmrob()` function, which builds the '`sparmodel`' object. We can do this easily by the available constructor function:

```
R> spar_glmrob <- constructor_sparmodel(name = "glmrob",
+    model_fun = model_glmrob)
```

For illustration purposes we contaminate 25% of `example_data2` above with outliers in the predictors and consider a binary version of the response variable `y`.

```
R> perc_cont <- 0.25
R> x <- example_data2$x;
R> set.seed(123)
R> np <- ncol(x) * nrow(x)
R> id_outliers_x <- sample(seq_len(np), perc_cont * np)
R> x[id_outliers_x] <- x[id_outliers_x] + 50
```

We can now estimate the SPAR algorithm with robust GLMs as marginal models and compare it to a version with marginal GLMs. We use no screening and `rp_gaussian()`. We set `measure = "mae"` i.e., we find the threshold ν by choosing the value which delivers the lowest mean absolute error. To improve stability in the estimates from `robustbase::glmrob`, we set the upper bound on goal dimension of the random projection to 25 (default would be $n/2 = 100$).

```
R> set.seed(1234)
R> spar_rob_res <- spar(x, ystar, family = binomial(),
+    model = spar_glmrob(), rp = rp_gaussian(msup = 25),
```

```
+    measure = "mae")
R> set.seed(1234)
R> spar_res <- spar(x, ystar, family = binomial(),
+    model = spar_glm(), rp = rp_gaussian(msup = 25),
+    measure = "mae")
R> spar_rob_res

spar object:
Smallest Validation Measure of 4.13e-01 reached for nummod=20,
              nu=1.94e-03 leading to 1000 / 1000 active predictors.
Summary of those non-zero coefficients:
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-1.171e-03 -2.430e-04  3.327e-05  3.458e-05  3.039e-04  1.266e-03

R> spar_res

spar object:
Smallest Validation Measure of 4.17e-01 reached for nummod=20,
              nu=1.81e-03 leading to 1000 / 1000 active predictors.
Summary of those non-zero coefficients:
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-1.085e-03 -2.311e-04  3.125e-05  3.531e-05  2.880e-04  1.133e-03
```

We observe that the attained `mae` is indeed lower for the robust version.

5. Conclusion

Package **spareg** can be employed for modeling high-dimensional data in a GLM framework, especially in settings where the number of predictors is much higher than the number of observations. The package provides an implementation of a computationally-efficient algorithm for sparse projected and average regression (SPAR), which combines variable screening and random projection in an ensemble of GLMs to reduce dimensionality of the predictors. Package **spareg** provides flexible classes for i) specifying the coefficient based on which screening should be performed (both in a classical fashion, where the predictors with the highest screening coefficient are selected for subsequent analysis or in a probabilistic fashion, where variables are sampled for inclusion with probabilities proportional to their screening coefficient), ii) generating the random projection to be employed in each marginal model, iii) specifying the marginal models to be used in the ensemble. Screening coefficients based on marginal correlation between the predictors and the response, marginal coefficients from a GLM or ridge coefficients are provided in the package. Moreover, several random projections are implemented: the Gaussian and sparse matrices which are data-agnostic and satisfy the JL lemma and the data-driven projection proposed in Parzer *et al.* (2024b) for linear regression and extended to GLMs in Parzer *et al.* (2024a). This data-driven approach, where information about the relationship among the responses and the predictors is incorporated in the random projection through a ridge coefficients, has the advantage of ensuring that the true regression coefficients can be recovered approximately after the projection. Methodologically, the SPAR algorithm with ridge screening coefficients and the data-driven random

projection (as proposed in Parzer *et al.* (2024a)) has been demonstrated to perform effectively in terms of predictive power and variable ranking in settings with correlated predictors and across different degrees of sparsity of the coefficient vector, making it a suitable method for high-dimensional settings with correlated predictors where the sparsity of the problem is unknown.

Moreover, the flexibility and adaptability of the **spareg** package in dealing with different types of screening, random projection and types of GLMs, make it an attractive choice for practitioners and researchers in a wide variety of data settings. It encourages exploration of new methods for variable screening and random projections or the combination of existing approaches to tailor solutions to specific data requirements.

Computational details

The results in this paper were obtained using R 4.5.0. R itself and all packages used are available from CRAN at https://CRAN.R-project.org/.

```
R> sessionInfo()

R version 4.5.0 (2025-04-11)
Platform: aarch64-apple-darwin20
Running under: macOS Sonoma 14.2.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: Europe/Vienna
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
[1] spareg_1.0.0  ggplot2_3.5.2

loaded via a namespace (and not attached):
 [1] generics_0.1.3          tidyr_1.3.1
 [3] robustbase_0.99-4-1     rstatix_0.7.2
 [5] shape_1.4.6.1           lattice_0.22-6
 [7] magrittr_2.0.3          grid_4.5.0
 [9] iterators_1.0.14        foreach_1.5.2
[11] glmnet_4.1-8            Matrix_1.7-3
```

```
[13] backports_1.5.0          Formula_1.2-5
[15] survival_3.8-3           energy_1.7-12
[17] purrr_1.0.4              scales_1.3.0
[19] codetools_0.2-20         abind_1.4-8
[21] Rdpack_2.6.4             cli_3.6.4
[23] expm_1.0-0               rlang_1.1.6
[25] rbibutils_2.3            gsl_2.1-8
[27] cowplot_1.1.3            munsell_0.5.1
[29] splines_4.5.0            withr_3.0.2
[31] VariableScreening_0.2.1 tools_4.5.0
[33] ggsignif_0.6.4           dplyr_1.1.4
[35] colorspace_2.1-1         ggpubr_0.6.0
[37] boot_1.3-31              ROCR_1.0-11
[39] broom_1.0.8              vctrs_0.6.5
[41] R6_2.6.1                 lifecycle_1.0.4
[43] car_3.1-3                MASS_7.3-65
[45] pkgconfig_2.0.3          gee_4.13-29
[47] pillar_1.10.2            gtable_0.3.6
[49] glue_1.8.0               Rcpp_1.0.14
[51] DEoptimR_1.1-3-1         tibble_3.2.1
[53] tidyselect_1.2.1         rstudioapi_0.17.1
[55] farver_2.1.2             carData_3.0-5
[57] labeling_0.4.3           compiler_4.5.0
```

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. doi:10.1137/060673096.

Bates D, Maechler M, Jagan M (2024). ***Matrix**: Sparse and Dense Matrix Classes and Methods*. doi:10.32614/CRAN.package.matrix. R package version 1.7-0.

Cannings TI, Samworth RJ (2017). "Random-Projection Ensemble Classification." *Journal of the Royal Statistical Society B*, **79**(4), 959–1035. doi:10.1111/rssb.12228.

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

Cantoni E, Ronchetti E (2001). "Robust Inference for Generalized Linear Models." *Journal of the American Statistical Association*, **96**(455), 1022–1030. doi:10.1198/016214501753209004.

Cheng X, Wang H, Zhu L, Zhong W, Zhou H (2024). ***MFSIS**: Model-Free Sure Independent Screening Procedures*. doi:10.32614/CRAN.package.mfsis. R package version 0.2.1.

Cho H, Fryzlewicz P (2012). "High Dimensional Variable Selection Via Tilting." *Journal of the Royal Statistical Society B*, **74**(3), 593–622. doi:10.1111/j.1467-9868.2011.01023.x.

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, pp. 81–90. Association for Computing Machinery, New York, NY, USA. ISBN 9781450320290. doi: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. doi:10.1080/01621459.2014.920256.

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. doi:10.1198/jasa.2011.tm09779.

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. doi:10.1214/10-imscoll606.

Fan J, Lv J (2007). "Sure Independence Screening for Ultra-High Dimensional Feature Space." *Journal of the Royal Statistical Society B*, **B 70**. doi:10.1111/j.1467-9868.2008.00674.x.

Fan J, Samworth R, Wu Y (2009). "Ultrahigh Dimensional Feature Selection: Beyond the Linear Model." *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.

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:10.1016/0095-8956(88)90043-3.

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

Johnson W, Lindenstrauss J (1984). "Extensions of Lipschitz Maps into a Hilbert Space." *Contemporary Mathematics*, **26**, 189–206. doi: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.

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, pp. 287–296. Association for Computing Machinery, New York, NY, USA. ISBN 1595933395. doi:10.1145/1150402.1150436.

Li R, Huang L, Dziak J (2022). ***VariableScreening**: High-Dimensional Screening for Semi-parametric Longitudinal Regression*. doi:10.32614/CRAN.package.variablescreening. R package version 0.2.1.

Li R, Zhong W, Zhu L (2012). "Feature Screening via Distance Correlation Learning." *Journal of the American Statistical Association*, **107**(499), 1129–1139. doi:10.1080/01621459.2012.695654.

Ma X, Zhang J (2016). "Robust Model-Free Feature Screening via Quantile Correlation." *Journal of Multivariate Analysis*, **143**, 472–480. doi:10.1016/j.jmva.2015.10.010.

Mai Q, Zou H (2013). "The Kolmogorov Filter for Variable Screening in High-Dimensional Binary Classification." *Biometrika*, **100**(1), 229–234. doi: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.

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

Microsoft, Weston S (2022). ***foreach**: Provides Foreach Looping Construct*. doi:10.32614/CRAN.package.foreach. R package version 1.5.2.

Microsoft Corporation, Weston S (2022). ***doParallel**: Foreach Parallel Adaptor for the **parallel** Package*. doi:10.32614/CRAN.package.doparallel. R package version 1.0.17.

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:10.1080/01621459.2019.1677240.

Parzer R, Filzmoser P, Vana-Gür L (2024a). "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.

Parzer R, Filzmoser P, Vana-Gür L (2024b). "Sparse Data-Driven Random Projection in Regression for High-Dimensional Data." *Technical Report 2312.00130*, arXiv.org E-Print Archive. doi:10.48550/arxiv.2312.00130.

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/.

Ryder N, Karnin Z, Liberty E (2019). "Asymmetric Random Projections." *Technical Report 1906.09489*, arXiv.org E-Print Archive. doi:10.48550/arxiv.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: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.

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

Tian Y, Feng Y (2021). ***RaSEn**: Random Subspace Ensemble Classification and Variable Screening*. doi:10.32614/CRAN.package.rasen. R package version 3.0.0.

Todorov V, Filzmoser P (2009). "An Object-Oriented Framework for Robust Multivariate Analysis." *Journal of Statistical Software*, **32**(3), 1–47. doi:10.18637/jss.v032.i03.

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

Vana-Gür L, Parzer R, Filzmoser P (2025). ***spareg**: Sparse Projected Averaged Regression in R*. doi:10.32614/CRAN.package.spareg. R package version 1.0.0.

Wang X, Leng C (2016). "High-Dimensional Ordinary Least-Squares Projection for Screening Variables." *Journal of the Royal Statistical Society B*, **78**, 589–611. doi: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. doi:10.1080/01621459.2014.993081.

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.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag. ISBN 978-3-319-24277-4. doi:10.1007/978-0-387-98141-3.

Xu C, Chen J (2014). "The Sparse MLE for Ultrahigh-Dimensional Feature Screening." *Journal of the American Statistical Association*, **109**(507), 1257–1269. doi:10.1080/01621459.2013.879531.

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. doi:10.1198/jasa.2011.tm10563.

Affiliation:

Laura Vana-Gür
Computational Statistics (CSTAT)
Institute of Statistics and Mathematical Methods in Economics
TU Wien
Wiedner Hauptstraße 8-10
1040 Vienna, Austria
E-mail: laura.vana.guer@tuwien.ac.at