RMIT University

Identifying Optimal Set of Pairwise Interaction Terms by SP-FSR Algorithm and Empirical Comparison with Other Methods

Author:
Yong Kai Wong

Supervisor:
Dr. Vural Aksakalli

A minor thesis submitted in partial fulfilment of the requirements for the Master of Statistics and Operations Research

in the

School of Science

June 17, 2018

Declaration

I certify that:

- the work of the thesis is my own, except where due acknowledgement has been made;
- no part of work in this minor thesis has been submitted previously to qualify for any other academic award;
- the content of the minor thesis is the result of work which has been carried out since the official enrollment date of the course;
- any editorial work carried out by a third party is acknowledged;
- the ethics procedures and guidelines have been followed;
- the work was supervised by Dr Vural Aksakalli.

Yong Kai Wong
June 17, 2018

Abstract

Some of the most commonly used models in supervised machine learning are linear and logistic regression models. One of the fundamental issues in these models is to select the best set of features to be included, particularly pairwise interaction terms. In this thesis, we develop a feature selection wrapper method called Two-Step SP-FSR for learning pairwise interactions. Given a supervised learning model and a specified model performance criterion, Two-Step SP-FSR first searches for an optimal set of main-effect features. Then it searches for relevant interaction features from these main effects while keeping the latter in training the learning model. Two-Step SP-FSR effectively enforces a strong hierarchy, meaning interaction features can be present only if both of its main-effect features are present. We run some computational experiments to compare Two-Step SP-FSR and other competitor methods, as well as the baseline models which are trained on full training sets, in order to detect interactions. Our competitor wrapper methods are Genetic Algorithm, Sequential Forward Selection and Sequential Floating Forward Selection. We also compare to one embedded method - `glinternet`. All competitor methods assume strong hierarchy. We evaluate their performances on test sets with root mean squared error and area under the curve (AUC) for regression and classification problems respectively. Two-Step SP-FSR outperforms all wrapper methods in each dataset. It results in a higher accuracy than the baseline learners, suggesting evidence of interactions. Though Two-Step SP-FSR does not always outperform `glinternet` in some experiments, it is more flexible since it can be extended to other learning models such as multinomial and Poisson regressions for interaction discovery.

Acknowledgements

I would first like to thank my thesis supervisor Dr. Vural Aksakalli for his motivation, continuous support and expert knowledge. Not only he was a dedicated supervisor, but also a role model. His wit and guidance have steered me in the right direction in my works and beyond. I have learnt a lot from him. I feel grateful to have him as my supervisor.

Besides Dr. Vural, I would like to express my gratitude to Dr. Babak Abbasi, Dr. Zeren D. Yenice, Dr. Alev Taskin Gumus and Mr Niranjan Adhikari for their previous works on the SP-FSR algorithm. Without their brilliant ideas and contributions, I would not have been even able to kick-start my thesis. I would also like to thank Dr. Haydar Demirhan for sharing his valuable experience in developing statistical software - one of the crucial ingredients in completing my minor thesis.

Last but not least, I am very thankful for my family and my partner - Lin Xin Kin for unyielding support throughout my years of study. Thank you.

Contents

List of Figures

List of Tables

1 Introduction

In supervised machine learning, identifying interactions can be crucial because interactions are scientifically meaningful and interesting in some applications. For instances, Schwender and Ickstadt (2008) show that interactions of single nucleotide polymorphisms play an important role in predicting cancer types. Hamilton (2011) suggests there exist interaction effects between political orientation and education levels, holding other background factors constant, in estimating the probability of seeing climate change as a threat.

Learning interactions is challenging because including interaction terms might cause overfitting issues and potentially compromise model accuracy performances. Worse, detecting interactions often fall into the high dimensionality curse where there are more features than the observations in the underlying dataset[1]. A common approach to reducing such high dimensionality problem is to apply feature extraction or feature selection methods.

A feature extraction method usually requires transforming feature space to lower dimensionality space, for example, Principal Component Analysis (PCA). However, they are not appropriate for learning interactions since feature transformation can cause loss of information and interpretability. A preferred alternative is feature selection. Feature selection can be loosely defined as determining an optimal set of descriptive features which is associated with the response (target) feature in a dataset by filtering out irrelevant or redundant features (Aksakalli and Malekipirbazari 2016).

There are three main categories of feature selection methods: filter methods, wrapper methods and embedded methods. Filter methods rely on statistical characteristics of data, such as distance and correlation measures, to eliminate poorly associated features. The popular filter methods include Minimum-redundancy-maximum-relevance (mRMR) (Peng, Long, and Ding 2005) and RELIEF (Sikonia and Kononenko 2003). Despite their computational efficiency, filter methods select relevant features by ignoring the performance of the underlying learning algorithm.

Wrapper methods search for the set of descriptive features which optimises a pre-specified accuracy performance measure given a learning model (Kohavi and John 1997). Despite a higher computational requirement, wrapper methods often produce more superior performance results. The widely used wrapper methods are Sequential Forward Selection (SFS), Sequential Backward Selection (SBS), and Genetic Algorithm (GA). An SFS method begins with an empty model (i.e. no descriptive features) and adds predictive features sequentially to create an optimal feature set whereas the latter begins with a full set (i.e. with all features) and eliminate one by one to achieve an optimal performance criterion.

In an iterative process, GA generates a population of candidate solutions where each candidate can be mutated or modified to form a new population to be evaluated in the next iteration based on a “fitness” or an objective value function (Siedlecki and Sklansky 2011).The GA is categorised as a meta-heuristic wrapper method. The meta-heuristic approaches often require

[1]Given p main effect features or variables, there are $\binom{p}{2}$ *pairwise* interaction terms. By including main effect features, the space of possible feature set grows to $p + \binom{p}{2} = \frac{p(p+1)}{2}$. The space becomes even larger when considering quadratic and higher-order terms.

even higher computational efforts and guarantee no optimal solutions due to their stochastic nature.

Embedded methods incorporate feature selection process when training the underlying learning algorithm, for instance, Least Absolute Shrinkage and Selection Operator (LASSO) (R. Tibshirani 1996). LASSO imposes penalty constraints on the coefficient estimation of a regression model. It determines the optimal feature set by shrinking the coefficient estimates of irrelevant features to zeroes. Its variations and extensions include net-elastic models (Friedman, Hastie, and Tibshirani 2010) and group variable selection method (Yuan and Lin 2006).

To apply feature selection methods in detecting interactions, a general approach is to extend the space of feature sets by including interactions themselves. However, direct applications might be prohibitive due to growing computational complexity. To alleviate the computational barrier, there has been much recent development in fitting models with the interactions using embedded methods (Simon and Tibshirani 2012; Lim and Hastie 2018; Bien, Taylor, and Tibshirani 2013; Shah 2016). By comparison, there has been little progress for filter and wrapper methods. A recent wrapper-based method is **S**tepwise c**O**ndtional likelihood variable selection for **D**iscriminant **A**nalysis (SODA) proposed by Li and Liu (2017). SODA is built on both forward and backward selections to select predictive main-effect and interactions features to optimise the extended Bayesian Information Criterion (EBIC). We are not aware of any recent development in filter methods for discovering interactions.

The objective of this study is to devise a wrapper method for learning interactions by extending Simultaneous Perturbation Stochastic Approximation in Feature Selection and Ranking (SP-FSR) algorithm introduced by Aksakalli and Malekipirbazari (2016). Given a learning model, this pseudo-gradient descent stochastic algorithm returns a set of predictive features which optimises a specified accuracy performance measure. We propose to utilise the SP-FSR algorithm because Yenice et al. (2018) and Aksakalli and Malekipirbazari (2016) empirically show that it can produce superior model performances compared to other wrapper methods based on the main-effect features. We wonder if it would yield excellent performances by including interaction features as well. Since we develop our method by applying the SP-FSR algorithm in two major steps, we call it “two-step SP-FSR”.

We run computational experiments to compare the performance of our method with other existing approaches. The experiment comprises two types of tasks: regression and binary classification. The target feature is a continuous variable in a regression task whereas it is binary in a classification task. In regression tasks, we compare our method with GA and sequential forward methods with the same learning model - a linear regression. We benchmark our method against an embedded method as well as a baseline learner, which is a model without any interaction term. While the wrapper methods can incorporate any learning algorithm, we decide to rely on the linear regression for comparability with the embedded method. Likewise, we set up a classification task in the exact methodology, except that we use a logistic regression as the learning model. In this study, we use the term “interaction” to refer all pairwise interaction features. We shall not consider quadratic and higher-order interaction terms.

The rest of this thesis paper is organised as follows. The next chapter presents important aspects of “interactions” in statistical context and how these aspects shape the design of our method. In Chapter 3, we review existing methods for interaction discovery and explain why we select some of them as competitors to two-step SP-FSR. Chapter 4 briefly discusses SP-FSR and delineates how we extend it to identify pairwise interactions. Chapter 5 describes our experimental setup and presents the empirical results. The last chapter concludes with a summary and highlights the direction of our future works.

2 Background

2.1 Notations

We use y_i and $x_{i,j}$ to denote the response and the value of j^{th} descriptive feature for i^{th} observation respectively. y_i and $x_{i,j}$ are also known as response and explanatory variables. Throughout this thesis, we use the term "feature" and shall not use them interchangeably with "variable", unless necessary, to be consistent with the context of feature selection. We use the upper cases to represent vectors and the boldface font for matrices. Suppose there are n observations, we write $Y = [Y_1, Y_2, ..., Y_n]^T$ and $X_j = [x_{1,h}, x_{2,j}, ...x_{n,j}]^T$ to denote n-vectors of responses and values of j^{th} features respectively. We also call X_j a main-effect feature.

Suppose there are p *main-effect* explanatory features, we define $\mathbf{X} = [x_{i,j}] \in \mathbb{M}^{n \times p}$ as the matrix of explanatory features for all observations. $\mathbf{X}$ can comprise of both continuous and categorical features; however, we shall not distinguish the feature types unless unnecessary. To represent a pairwise interaction between two descriptive features, say k^{th} and j^{th} features, we express:

$$X_{k:j} = X_k \times X_j = [x_{1,k} \times x_{1,j}, x_{2,k} \times x_{2,j}, ..., x_{n,k} \times x_{n,j}]^T \text{ for } k \neq j \tag{1}$$

Loosely following notations of generalised linear models (Bilder 2015), we write a linear predictor function or a link function as follow:

$$f(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = \beta_0 + \sum_{j=1}^{p} \beta_j X_j \tag{2}$$

The β_0, β_1, ... β_p are known as regression coefficients. Note that Equation 2 contains no error term since a link function describes how the mean of Y is related to the linear combination of the explanatory features via a mathematical function. In a standard linear regression, $f(Y)$ is an identity function:

$$\mathbb{E}(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p \tag{3}$$

$\mathbb{E}(Y)$ is the expectation of Y. Equation 3 is appropriate for a continuous response. Logistic regression model is most commonly used for a binary response i.e. $Y_i \in \{0, 1\}$, The link function of a logistic regression model is $f(\cdot) = \exp(\cdot)/(1 + \exp(\cdot))$. Therefore, $\mathbb{E}(Y)$ can, therefore, be written as:

$$\mathbb{E}(Y) = \frac{\exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p)}{1 + \exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p)} \tag{4}$$

To include pairwise interaction features in a link function, we write:

$$f(Y) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j + \sum_{j<k} \beta_{j:k} X_{j:k} \tag{5}$$

$\beta_{j:k}$ is the interaction coefficient between k^{th} and j^{th} features. Equation 5 is the basic model formulation in this study.

2.2 Aspects of Interactions

Cox (1984) provides a statistical exposition of various aspects of interactions on the structure of $\mathbb{E}(Y)$. He states a precise definition of "interactions": given two features X_j and X_k, when a function $f(X_j, X_k)$ cannot be expressed as $h_1(X_k) + h_2(X_j)$ for some functions h_1 and h_2, there exists an interaction in f between X_k and X_k. Cox further classifies $\mathbf{X}$ into *treatment*, *intrinsic*, and *non-specific* features (variables).

The treatment features refer to control factors in observational studies. The intrinsic features are factors beyond the control of experimenters, for examples, age and gender. The non-specific features are known as random factors such as experimental blocks and replicates. By restricting to two-way interactions, Cox argues there are three different kinds to be considered:

- Treatment $\times$ Treatment;
- Treatment $\times$ Intrinsic;
- Treatment $\times$ Non-specific.

In our study, we do not distinguish these kinds above for pragmatic reasons. Also, Cox ignores the case of Intrinsic $\times$ Intrinsic, which can be of interest. For example, there might be an interaction effect between gender and age in estimating an individual's body fat mass.

With a careful formulation, Cox asserts that F-tests are powerful to detect interactions of interest whereas graphical methods are appropriate for situations where the interaction cannot be completely estimated. Cox's methodology has a few shortcomings. First, F-tests or other statistical hypothesis tests might be difficult to formulate, and graphical methods might be infeasible when there are many features. Second, statistical tests can severely limit the scope of how significant main-effect and interaction features can be selected. For example, models $f(Y) = \beta_0 + \beta_1 X_1 + \beta X_2$ and $f(Y) = \beta_0 + \beta_1 X_1 + \beta X_2 + \beta_{1:2} X_{1:2}$ can be compared via Analysis of Variance (ANOVA) to assess if the interaction term of $X_{1:2}$ should be included. However, we cannot *easily* compare models $f(Y) = \beta_0 + \beta_1 X_1 + \beta X_2 + \beta_{1:2} X_{1:2}$ and $f(Y) = \beta_0 + \beta_1 X_4 + \beta X_5 + \beta_{4:5} X_{4:5}$ in a nested framework. Feature selection hence becomes useful to compare different model specifications.

When interaction features are included, a typical feature selection process might lead to an outcome where interaction features are selected but their main effects are not. Since it is rarely to have interactions without main effects, Lim and Hastie (2018) posit to enforce some degree of hierarchy on model specification. They define a model to follow a strong hierarchy when an interaction is present only if both of its main effects are present too. A model is weakly hierarchical as long as either of the main effect features are present. Anti-hierarchical model is when interactions are only pairs of main effects are absent. A pure-interaction model contains no main-effect features. While the true structure of hierarchy is unknown, anti-hierarchical and pure-interaction forms are rare. In Chapter 4, we shall explain how we develop Two-Step SP-FSR by assuming a strong hierarchy.

3 Existing Approaches

In this chapter, we briefly discuss some wrapper methods and an embedded method used to discover interactions. We explain why we choose some of these methods in our computational experiments.

3.1 Wrapper Methods

As mentioned in Chapter 1, wrapper methods are applied to the expanded set of main-effect and interaction features. As a result, the strong hierarchy might not hold. To address this shortcoming, a solution is first to identify both relevant main and interaction effects, and then include the omitted main-effect features from the identified interactions into the selection. Wrapper methods which can incorporate this approach are SFS, SFFS and GA. SBS or SBFS are ruled out because they require $p < n$ so that the backward elimination can work.

SFFS performs better than SFS since it validates the possibility of improvement of the performance criterion when some feature is excluded (Pudil et al. 1994). However, as we shall show in Chapter 5, we include SFS as an alternative to SFFS since the latter runs into a stall in some experiments. We also do not include SODA (Li and Liu 2017) in comparison since this method is restricted with EBIC whereas our experiments require other evaluation measures. As a result, we choose SFS, SFFS and GA as our competitor wrapper methods.

3.2 Embedded Method

Lim and Hastie (2018) develop **g**roup-**l**asso **inter**action **net**work (`glinternet`) by imposing a penalty function $p(\beta)$ on the coefficients when minimising the log function, $\mathcal{L}(Y;\beta)$ as follows:

$$\arg\min_{\beta} \mathcal{L}(Y;\beta) + \lambda p(\beta) \tag{6}$$

λ is the tuning parameter where $\lambda > 0$. λ can be determined via a grid search with cross validations. If Y is continuous, the loss function is a squared-error loss:

$$\mathcal{L}(Y;\beta) = \frac{1}{2}\Bigg[Y - \Big(\beta_0 + \sum_{j=1}^{p}\beta_j X_j + \sum_{j<k}\beta_{j:k}X_{j:k}\Big)\Bigg]^2$$

For a binary response, the loss function is given by:

$$\mathcal{L}(Y;\beta) = -\Bigg[Y^T\Big(\beta_0 + \sum_{j=1}^{p}\beta_j X_j + \sum_{j<k}\beta_{j:k}X_{j:k}\Big) - \mathbf{1}^T \log\Big(\mathbf{1} + \exp(\beta_0 + \sum_{j=1}^{p}\beta_j X_j + \sum_{j<k}\beta_{j:k}X_{j:k})\Big)\Bigg]$$

1 is an n-vector of ones. The penalty function specification depends on the feature types of **X**. For example, if all (main-effects) explanatory features are continuous, $p(\beta)$ becomes:

$$p(\beta) = \sum_{j=1}^{p} |\beta_j| + \sum_{j<k} ||\beta_{j:k}||_2$$

$|\cdot|$ and $||\cdot||_2$ are absolute value and Euclidean norms. For more details, Lim and Hastie (2018) provide more theoretical explanations, including penalty specifications for interactions between categorical and continuous explanatory features, and interactions among categorical explanatory features as well. The authors developed an extension library (package) for `glinternet` (2018) in `R` - a statistical computing language.

4 Identifying Interactions with SP-FSR Algorithm

4.1 A Primer in SP-FSR Algorithm

SP-FSR is built on Simultaneous Perturbation Stochastic Approximation (SPSA) developed by J. Spall (1992). Here, we first outline SPSA and then describe how Aksakalli and Malekipirbazari (2016) apply the binary version of SPSA (BSPSA) (J. Spall and Qi 2011) for feature selection. Lastly, we summarise how Yenice et al. (2018) improve BSPSA using non-monotone Barzilai & Borwein (BB) search method.

Let $\mathcal{L} : \mathbb{R}^p \mapsto \mathbb{R}$ be a loss function whose functional form is not explicitly known, but its noise measurement can be observed as:

$$y(w) := \mathcal{L}(w) + \varepsilon(w) \tag{7}$$

ε is the noise, y is the noise measurement and $w \in D \subset \mathbb{R}^p$. Let $g(w) := \nabla\mathcal{L} = \frac{\partial \mathcal{L}}{\partial w}$ be the gradient of $\mathcal{L}$. Starting with a random guess $\hat{w}_0$, SPSA moves toward the optimal solution w^* recursively via:

$$\hat{w}_{k+1} := \hat{w}_k - a_k \hat{g}(\hat{w}_k)$$

$a_k \geq 0$ is an iteration gain sequence whereas $\hat{g}(\hat{w}_k)$ is the approximate gradient at $\hat{w}_k$ at iteration k. At each k, $\hat{w}_k$ is perturbed simultaneously by random offsets, $\pm c_k \Delta_k$ generated from a predetermined symmetric distribution. c_k is a non-negative gradient gain sequence while Δ_k is known as the simultaneous perturbation vector. $\{\Delta_k\}_{k=1}$ must be a mutually independent sequence and also independent of $\{\hat{w}_k\}_{k=0}$. A Bernouli distribution with zero mean is usually used to generate Δ_k. The simultaneous perturbations around $\hat{w}_k$ is given by:

$$\hat{w}^{\pm}_{k+1} := \hat{w}_k \pm c_k \Delta_k$$

At iteration k, the noisy measurements of $\hat{w}^{\pm}_k$ become:

$$y^+_k := \mathcal{L}(\hat{w}_k + c_k\Delta_k) + \varepsilon^+_k$$
$$y^-_k := \mathcal{L}(\hat{w}_k - c_k\Delta_k) + \varepsilon^-_k$$

Subsequently,

$$\hat{g}_k(\hat{w}_k) : = \left[\frac{y^+_k - y^-_k}{w^+_{k1} - w^-_{k1}}, ..., \frac{y^+_k - y^-_k}{w^+_{kp} - w^-_{kp}}\right]^T = \frac{y^+_k - y^-_k}{2c_k}[\Delta^{-1}_{k1}, ..., \Delta^{-1}_{kp}]^T$$

y_k^+ and y_k^- are used to approximate the gradient. $y(\hat{w}_{k+1})$ is used to compute the performance of iteration $k+1$. J. C. Spall (2003) offers more theoretical elaboration about SPSA, which we shall not proceed further. In a binary version of SPSA (BSPSA), the domain of the loss function becomes $D = \{0,1\}^p$ and the gain sequence c_k is constant, say $c_k = c$ whereas $\hat{w}_k^\pm$ are bounded and rounded before evaluating $y_k^\pm$.

Aksakalli and Malekipirbazari (2016) formulate a feature selection as follow. Given a nonempty feature subset $\mathbf{X}' \subset \mathbf{X}$, we define $\mathcal{L}_{\mathbb{M}}(\mathbf{X}', Y)$ as the true but unknown value of performance criterion for a wrapper model, $\mathbb{M}$. As we shall show in next chapter, we use linear and logistic regression models as wrappers in our computational experiment. We train $\mathbb{M}$ by evaluating a specified error measure, $y_{\mathbb{M}}(\mathbf{X}', Y)$. The error measures include mean squared error (MSE) for regression task and misclassification error rate for classifier model. In BSPSA, the error measure is the noisy measurement function. Equation 7 hence can be expressed as $y_{\mathbb{M}} = \mathcal{L}_{\mathbb{M}} + \varepsilon$. Essentially, the wrapper feature selection problem aims to determine an optimal feature subset $\mathbf{X}^*$:

$$\mathbf{X}^* := \arg \min_{\mathbf{X}' \subset \mathbf{X}} y_{\mathbb{M}}(\mathbf{X}', Y) \tag{8}$$

Equation 8 is the basis of the initial version of SP-FSR. Using a non-monotone Barzilai & Borwein search method (Barzilai and Borwein 1988), Yenice et al. (2018) speed up the convergence of rate of Equation 8 by losing a mimimal amout of accuracy performance (based on the pre-specified error measure). We shall not discuss the Barzilai & Borwein method further, but highlight the important modifications proposed by Yenice et al. (2018). Yenice et al. (2018) estimate the gain sequence by:

$$\hat{a}_k = \frac{\nabla \hat{w}_k^T \nabla \hat{g}(\hat{w}_k)}{\nabla \hat{g}^T(\hat{w}_k) \nabla \hat{g}(\hat{w}_k)} \tag{9}$$

To ensure its non-negativity, a closed boundary is imposed on the current gain (Equation 9):

$$\hat{a}_k^{'} = \max\{a_{\min}, \min\{\hat{a}_k, a_{\max}\}\}$$

$a_{\min}$ and $a_{\max}$ are the minimum and the maximum of gain sequence $\{\hat{a}_k\}_k$ at the current iteration k. Compared to the initial version, SP-FSR smooths the gains by averaging their values at the current and last two iterations, leading in a decrease in coverage time. To reduce distortion in convergence direction, the current $\hat{g}_k(\hat{w}_k)$ and its m predecessors are averaged.

In summary, SP-FSR is refined to converge much faster than the original version proposed by Aksakalli and Malekipirbazari (2016), at a small expense in the loss function. Algorithm 1 summarises the pseudo code of SP-FSR which is adapted from Yenice et al. (2018). Note that it was previously known as "SPSA-FS" but it was renamed as "SP-FSR". SP-FSR has no stopping rule; hence it requires the maximum number of iterations, i.e. M in Algorithm 1.

SP-FSR requires to fine-tune the initial solution $\hat{w}_0$ and gradient gain constant c. Aksakalli and Malekipirbazari (2016) recommend $\hat{w}_0 = [0.5, 0.5, ..., 0.5]^T$ and $c = 0.05$.

Aksakalli, Abbasi, and Wong (2018) develop an extension library (package) to implement the SP-FSR algorithm in R. This package is called spFSR. We implement Two-Step SP-FSR with this package.

Algorithm 1 SP-FSR Algorithm

```
1: procedure SP-FSR($\hat{w}_0$, $c$, $M$)
2: Initialise $k = 0$, $m = 0$
3: do:
4:     Simulate $\Delta_{k,j} \sim \text{Bernoulli}(-1, +1)$ with $\mathbb{P}(\Delta_{k,j} = 1) = \mathbb{P}(\Delta_{k,j} = -1) = 0.5$
5:     $\hat{w}_k^{\pm} = \hat{w}_k \pm c\Delta_k$
6:     $\hat{w}_k^{\pm} = B(\hat{w}_k^{\pm})$        ▷ $B(\bullet)$ = component-wise $[0, 1]$ operator
7:     $\hat{w}_k^{\pm} = R(\hat{w}_k^{\pm})$        ▷ $R(\bullet)$ = component-wise rounding operator
8:     $y_k^{\pm} = \mathcal{L}(\hat{w}_k \pm c_k\Delta_k) \pm \varepsilon_k^{\pm}$
9:     $\hat{g}_k(\hat{w}_k) = \left(\frac{y_k^+ - y_k^-}{2c}\right)[\Delta_{k1}^{-1}, ..., \Delta_{kp}^{-1}]^T$        ▷ $\hat{g}_k(\hat{w}_k)$ = the gradient estimate
10:    $\hat{g}_k(\hat{w}_k) = \frac{1}{m+1}\sum_{n=k-m}^{k} \hat{g}_n(\hat{w}_k)$        ▷ Gradient Averaging
11:    $\hat{a}_k = \frac{\nabla \hat{w}^T \nabla \hat{g}(\hat{w})}{\nabla \hat{g}^T(\hat{w}) \nabla \hat{g}(\hat{w})}$        ▷ $\hat{a}_k$ = BB Step Size
12:    if $\hat{a}_k < 0$ then
13:        $\hat{a}_k = \max\left(\min\{\hat{a}_k\}, \min\{\hat{a}_k, \max\{\hat{a}_k\}\}\right)$
14:    $\hat{a}_k = \frac{1}{t+1}\sum_{n=k-t}^{k} \hat{a}_n$ for $t = \min\{2, k\}$        ▷ Gain Smoothing
15:    $\hat{w}_k^{\pm} = \hat{w}_k \pm a_k \hat{g}_k(\hat{w}_k)$
16:    $k = k + 1$, $m = k$
17: while $(k < M)$
18: Output: $\hat{w}_M^{\pm} = R(\hat{w}_M^{\pm})$
```

4.2 Two-Step SP-FSR for Learning Interactions

Two-Step SP-FSR runs SP-FSR in two main steps. In the first step, given a specified error measure and a wrapper model, it selects an optimal set of main-effect features $\mathbf{X}' \in \mathbf{X}$. Then it creates a candidate set of pairwise interactions among the selected main-effect features. Let $\mathbf{Z}$ be the candidate set. In the second step, SP-FSR is run to search predictive interaction features from $\mathbf{Z}$ while keeping the significant main-effect features $\mathbf{X}'$ in training the wrapper. The search method effectively enforces a strong hierarchy: an interaction is present only if both of its main effect features are present. The outcome is a feature set of $[\mathbf{X}', \mathbf{Z}']$ where $\mathbf{Z}' \in \mathbf{Z}$ is the optimal interaction feature subset from Step 2.

Note that SP-FSR requires the number of features to be selected. Otherwise, SP-FSR will decide automatically by leveraging its stochastic optimisation structure. Two-Step SP-FSR inherits this semi-heuristic nature of SP-FSR. Therefore, the number of features selected can be determined via a grid search or an automatical mode at each step. To reduce the computational complexity, we prefer the latter approach in Two-Step SP-FSR. Similar to SP-FSR, Two-Step SP-FSR requires to specify maximum iterations, gain sequence and initial solution. Algorithm 2 summarises the pseudo code of Two-Step SP-FSR.

Algorithm 2 Two-Step SP-FSR Algorithm

```
1: procedure Two-Step SP-FSR($\hat{w}_0$, $c$, $M$, $\mathbf{X}$)
2: Run SP-FSR($\hat{w}_0$, $c$, $M$) to yield $\mathbf{X}' \in \mathbf{X}$
3: Create candidate set $\mathbf{Z}$ from $\mathbf{X}'$
4: Keep $\mathbf{X}'$ and run SP-FSR($\hat{w}_0$, $c$, $M$) to yield $\mathbf{Z}' \in \mathbf{Z}$
5: Output: $[\mathbf{X}', \mathbf{Z}']$
```

Unlike the embedded method such as `glinternet`, Two-Step SP-FSR can compare the wrapper performances between two stages. If the first stage yields a more superior performance than the second stage, it indicates an overidentification issue, implying there might be no true interaction features. We do not strictly enforce any overidentification rule in Two-Step SP-FSR. Instead, we store both performance results so that practitioners can decide to include interactions.

Let's illustrate the concept of Two-Step SP-FSR (see Algorithm 2) with the following simple example. Suppose we have a dataset of four explanatory features, $[Y, X_1, X_2, X_3, X_4]$ where Y is continuous. Let use a simple linear regression as the wrapper and evaluate its with MSE. Two-Step SP-FSR discovers interactions from $[X_1, X_2, X_3, X_4]$ in the following procedure:

1. Run SP-FSR and say it selects $\mathbf{X}' = \{X_1, X_2, X_3\}$ as the optimal main effects.
2. Among $\mathbf{X}'$, the candidate set of pairwise interaction features is generated. Since three main-effect features are selected, the candidate set contains $\binom{3}{2} = 3$ pairwise interaction features:

$$\mathbf{Z} = \{X_{1:2}, X_{1:3}, X_{2:3}\}$$

3. By keeping $\mathbf{X}'$, run SP-FSR to search for the optimal interaction feature subset $\mathbf{Z}' \in \mathbf{Z}$. Say $\mathbf{Z}' = \{X_{1:2}, X_{1:3}\}$.
4. The final set of both main-effect and interaction features is:

$$[\mathbf{X}', \mathbf{Z}'] = \{X_1, X_2, X_3, X_{1:2}, X_{1:3}\}$$

5. (Optional) The trained linear regression model (link function) hence becomes:

$$\mathbb{E}(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_{1:2} X_{1:2} + \beta_{1:3} X_{1:3}$$

5 Experimental Studies

5.1 Experimental Setup

We aim to compare the performance of two-step SP-FSR on regression and classificaton problems. As mentioned in Chapter 3, the competitor methods are `glinternet`, SFFS, SFS and GA as well as the baseline learner. For the wrapper methods SFFS, SFS and GA, we use the same learning or wrapper model. `glinternet` has the similar model formulation, except that it imposes a penalty function on regression coefficients (see Equation 6). For regression problems, the learning model is a simple linear regression (see Equation 3) whereas it is a logistic regression model for classification problems (see Equation 4). To compare with two-step SP-FSR and `glinternet`, we impose a strong hierarchy on each wrapper method meaning corresponding main effects must be present for every interaction feature.

In our experiment, we split the underlying dataset into a training set and a test set by a ratio of 80:20 in each problem. We train the learning model with each feature selection method on the training set. During the feature selection process, we subsample the training set with a specified number of cross validations. We exempt the baseline learners from subsampling and feature selection. In other words, the baseline learners run on the full *training* set with all main effects only. Then we cast predictions on the test set with the train model from each feature selection method. Finally, by benchmarking against the baseline learners, we evaluate the performance of two-step SP-FSR and its competitors on the test set based on some error measures. For reproducibility, we run the experiment with a set of different random seeds.

While commonly used, we do not use information criteria, such as Akaike's Information Criterion (AIC) and Schwarz's Bayesian Criterion (BIC), in the feature selection procedure. Since information criteria rely on n, we opine that they are not appropriate because each sub-training set might have different number of observations during subsampling. Following the wrapper approach described by Kohavi and John (1997), we use the error measures for performance benchmarking in the feature selection instead.

For regression tasks, we use root mean squared error (RMSE) for evaluation. RMSE is specific to the unit measurement of underlying Y; it is not biased upward like R-squared measure. A lower value RMSE indicates a higher accuracy rate. For binary classification tasks, we evaluate the methods using the area under the receiver operating characteristic (ROC) curve, or AUC. Loosely speaking, AUC measures the area of a plot True Positive Rate (TPR) against False Positive Rate (FPR). A logistic regression model returns probabilistic predictions and the predictions are classified into 0-1 response given a threshold level. We set the threshold at 0.5 which is used to compute TPR and FPR. A higher value of AUC implies a higher accuracy rate.

Table 5.1 lists the datasets for regression and classification tasks. These datasets are taken from UCI Machine Learning Repository (Lichman 2013) and DCC Regression DataSets (Torgo 2017). Each dataset's explanatory features are continuous. For regression tasks, we apply 10 cross validations on their training sets, except Boston Housing due to its small size. We use 4 cross validations instead. For each classification task, we apply stratified 5-cross validation sampling. Figure 5.1 depicts our experimental setup. On a computer with a processor of 2.3

Table 5.1: Datasets for experiments. p represents the number of explanatory features excluding exclude the response variables and identifier attributes; n denotes the number of observations.

Task	Dataset	p	n	Source
Regression	Ailerons	39	13750	DCC
	Elevators	17	16559	DCC
	Boston Housing	13	506	UCI
Classification	Sonar	9	214	UCI
	Ionosphere	34	351	UCI

GHz Intel Core i5, we run the experiment in `R` (Version 3.4.2) with the following packages:

- `glinternet` (Lim and Hastie 2018, Version 1.0.7) for hierarchical group-lasso regularization;
- `spFSR` (Aksakalli, Abbasi, and Wong 2018, Version 1.0.0) to run Two-Step SP-FSR; and
- `mlr` packages (Bischl et al. 2016, Version 2.11) to implement SFS, SFFS and GA.

The `R` scripts are explained further and presented in Appendix.

5.2 Regression Task Results

Table 5.2 presents the empirical results of regression tasks. In this table, p_0 and p_1 represent the number of main-effect features selected and the number of interaction features respectively. p_1 does not apply to the baseline learner. Due to instability of SFFS implementation in `R`, we replace it with SFS for regression tasks. For a more reliable comparison, we evaluate each method based on their test RMSE values.

For Ailerons dataset, Two-Step SP-FSR reduces the test RMSE from the baseline value of 0.0179789 by approximately 5.8 % to 0.0169259. However, it comes to the second as `glinternet` yields the lowest RMSE at 0.0167777. GA and SFS underperform, especially GA which produces a higher test RMSE. As expected, SFS tends to choose a smaller subset of main-effect features and hence a smaller size of interaction. On average, Two-Step SP-FSR, `glinternet` and GA choose around 18 to 25 main-effect features. Two-Step SP-FSR results in a higher number of interactions ($p_1 \approx 120$), which is twice as those of `glinternet` and GA.

For Boston Housing dataset, all methods beat the baseline learner. Two-Step SP-FSR outperforms other wrapper methods and reduces the baseline test RMSE by around 5.5 %. However, `glinternet` yields a superior result - an RMSE of 3.371138 down from the baseline value of 3.959734. On average, all methods select at least 9 out of 13 main-effect features where `glinternet` almost selects all ($p_0 \approx 12.9$). The number of interaction features varies such that `glinternet` results in the largest feature set.

All methods beat the baseline learner as well in Elevator dataset. Two-Step SP-FSR leads other competitors with the lowest test RMSE at 0.2422305 equivalent to a reduction of 17.7 % from the baseline value. `glinternet` underperforms GA by a slight margin. Except for

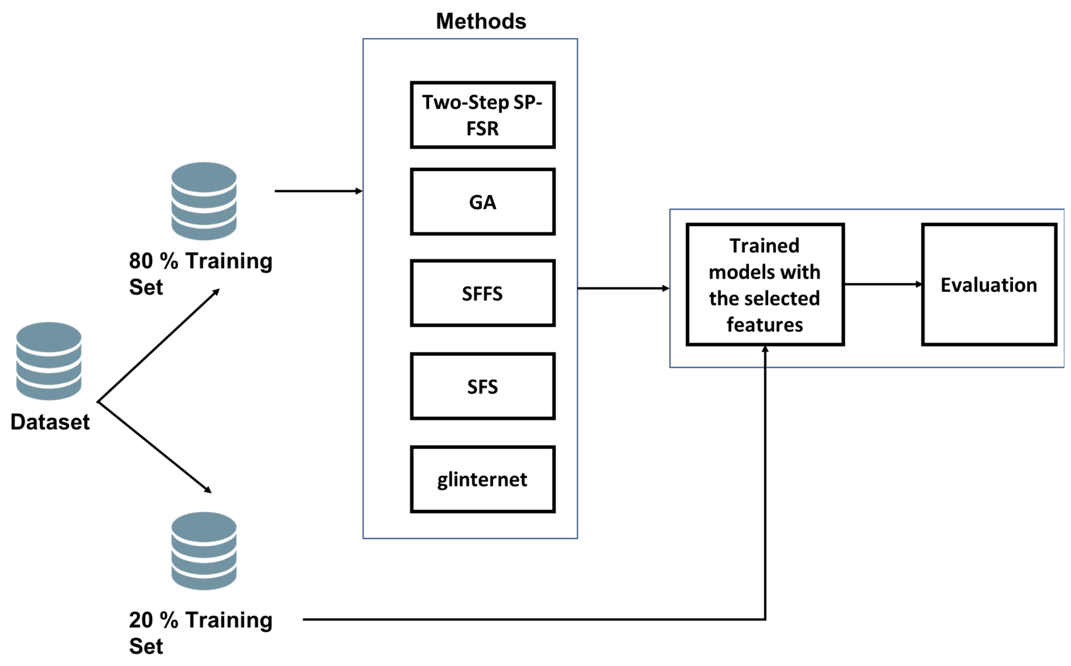

Figure 5.1: Experimental Setup. The dataset is split into training and test sets by 80/20 ratio. The learning model is estimated based on the training set via various feature selection methods. The trained model is used to make predictions on the test sets. The test predictions are used to evaluate the performance of each method

Table 5.2: Empirical comparison of methods on regression tasks. p_0 represents the number of main-effect features selected. p_1 represents the number of interaction features selected. p_1 does not apply to the baseline learner. p_0 and p_1 are rounded to two nearest decimals. GA and SFS are abbreviations of Genetic Algorithm and Sequential Forward Selection respectively. The best method is bolded in each dataset.

Dataset	Method	Mean Test RMSE	Mean Train RMSE	p_0	p_1
Ailerons	Baseline	0.0179789	0.0171819	40.00	NA
	Two-Step SP-FSR	0.0169259	0.0157097	21.33	127.33
	glinternet	**0.0167777**	**0.0157796**	**25.00**	**64.00**
	GA	0.0198347	0.0156096	17.67	65.33
	SFS	0.0173200	0.0165483	4.00	3.00
Boston Housing	Baseline	3.959734	4.869756	13.00	NA
	Two-Step SP-FSR	3.739827	3.277893	9.92	32.67
	glinternet	**3.371138**	**3.106602**	**12.92**	**65.00**
	GA	3.840331	3.143898	10.75	27.75
	SFS	3.864182	3.334595	11.00	19.33
Elevators	Baseline	0.2943325	0.2890896	18.00	NA
	Two-Step SP-FSR	**0.2422305**	**0.2394952**	**14.67**	**73.00**
	glinternet	0.2506703	0.2485003	16.00	42.00
	GA	0.2446648	0.2437792	12.33	43.00
	SFS	0.2526974	0.2523360	7.00	3.00

SFS, all methods choose at least 12 out of 18 main features. Two-Step SP-FSR produces the largest feature subset on average ($p_1 \approx 73$).

5.3 Classification Task Results

Table 5.3 presents the empirical results of classification tasks. Unlike regression tasks, we encounter no issues in implementing SFFS on the classification datasets. Similarly, we evaluate each method based on their test AUC values. For Sonar dataset, all methods as well the baseline logistic regression result in a perfect train AUC at 1. The exception is SFFS though its train AUC is as high as 0.96. Their test AUC values decline sharply below 0.7, suggesting all methods cause overfitting issues. This can be due to the small dataset. Two-Step SP-FSR yields a test AUC of 0.6810516 which is 12 % above the baseline value of 0.6059524. Although it tops other wrapper methods, it underperforms `glinternet` whose test AUC is 0.736.

Similarly, for Ionosphere dataset, Two-Step SP-FSR is the best wrapper method whereas GA and SFFS yield AUC values worse than the baseline logistic regression. Two-Step SP-FSR ekes out an AUC of 0.84 - not significantly higher than the baseline whereas `glinternet` has a superior performance with an AUC above 0.90. In both datasets, `glinternet` selects the most number of main-effect features and consequently the most number of interactions on average. SP-FSR tends to select smaller feature subsets compared to GA but larger compared to SFFS.

Table 5.3: Empirical comparison of methods on classification tasks. p_0 represents the number of main-effect features selected. p_1 represents the number of interaction features selected. p_1 does not apply to the baseline learner. p_0 and p_1 are rounded to two nearest decimals. GA and SFFS are abbreviations of Genetic Algorithm and Sequential Floating Forward Selection respectively. The best method is bolded in each dataset.

Dataset	Method	Mean Test AUC	Mean Train AUC	p_0	p_1
Sonar	Baseline	0.6059524	1.0000000	60	NA
	Two-Step SP-FSR	0.6810516	1.0000000	16.42	65.50
	glinternet	**0.7367063**	**1.0000000**	**45.75**	**82.83**
	GA	0.6117063	1.0000000	22.00	110.42
	SFFS	0.6257937	0.9647436	11.67	7.17
Ionosphere	Baseline	0.8294573	0.9973914	33	NA
	Two-Step SP-FSR	0.8432386	0.9797910	12.083	28.42
	glinternet	**0.9109963**	**0.9913929**	**27.17**	**37.83**
	GA	0.7678366	0.9813844	15.83	59.42
	SFFS	0.8207005	0.9580975	8.50	5.67

6 Conclusion and Future Works

Two-Step SP-FSR yields a more accurate performance among all wrapper methods. We opine that its excellent results are attributed by SP-FSR which is superior in selecting optimal sets of main effect features. Although it does not outperform `glinternet` in most of the empirical problems, we conjecture that Two-Step SP-FSR's performances are consistent under the presence of strong hierarchy. `glinternet` determines its key parameter λ via a grid search of at least 50 different values; hence, it can learn better than Two-Step SP-FSR. Ideally, we could apply the grid search on p_0 and p_1 for Two-Step SP-FSR as well, but it would cause higher computational costs which is a typical tradeoff among wrapper methods.

However, Two-Step SP-FSR is more flexible than `glinternet` since it can incorporate any learning model, such as multinomial, Poisson, and ordinal regression models. Besides, `glinternet` does not have a mechanism to address overidentification issue, i.e. there exist no true interaction features. An alternative is run LASSO to benchmark against `glinternet`, with no guarantee if both will select the same set of optimal main-effect features. Two-Step SP-FSR has a competitive edge over `glinternet` as the former keeps the same set of main-effect features in searching for the pairwise interactions. Two-Step SP-FSR is a wrapper method and hence non-parametric. Unlike `glinternet`, it can be easily modified to include quadratic and higher-order interactions terms. In future works, we will incorporate other models and higher-order terms with Two-Step SP-FSR.

Though our experiments show that interactions are likely to exist, we suggest running simulation studies for other hierarchical scenarios. The simulation studies will include true strong, weak, anti-hierarchical, pure-interaction, and no-interaction types. So far our experiments in this study focus on real datasets with $n > p$ and all explanatory features are numeric. Therefore, we will also test on datasets where $p \gg n$ and comprise mixed feature types.

Appendix I

The R codes are structured to mirror the experimental setup (see Figure 5.1). The source codes are divided into the following self-explanatory scripts. These scripts read and train the models on the training sets with various feature selection methods and evaluate their performances on the test sets in an automatical fashion.

R Codes for Regression Tasks

Baseline Learner for Regression

```r
# Baseline learners without interactions: regression tasks ----
# Compare baseline learners with other methods
# 0. Load packages ----
library(mlr)
library(tidyverse)

# 1. Run baseline learners ----
# Initialise the data frame to store results,
# Prepare data sets and target continuous features
result      <- data.frame()

data_files  <- c('BostonHousing', 'Ailerons', 'Elevators')
targets     <- c('medv', 'V41', 'V19')

# Common learner: linear regression
wrapper     <- makeLearner('regr.lm')

# Loop for each dataset
for(j in 1:length(data_files) ){

  train_data  <- read.csv(paste0(data_files[j], '_train.csv'))
  test_data   <- read.csv(paste0(data_files[j],'_test.csv'))

  train_task  <- makeRegrTask(data = train_data, target = targets[j])
  test_task   <- makeRegrTask(data = test_data, target = targets[j])

  baseModel   <- train(wrapper, train_task)
  train_pred  <- predict(baseModel, train_task)
  test_pred   <- predict(baseModel, test_task)

  result[j, 'Data'] <- data_files[j]
  result[j, 'AIC']  <- AIC(baseModel$learner.model)
  result[j, 'BIC']  <- BIC(baseModel$learner.model)
  result[j, 'Train_RMSE'] <- mlr::performance(train_pred, mlr::rmse)
  result[j, 'Test_RMSE']  <- mlr::performance(test_pred, mlr::rmse)

}

# Save the result as a csv file
write.csv(result, 'baselearner_regr.csv', row.names = FALSE)
```

Modified Make Regression Task Function

```r
#' @description Generating pairwise interaction terms of a data and
# '              pass in to a regression task
#'
#' @param data A data frame containing the features and target variable(s)
#' @param target Name of the target variable
#' @param order Interaction order. The allowed values are 1, 2, and 3.
#'              Default is 2L

modifiedMakeRegrTask <- function(data, target, order = 2,
                                  remove.constant = TRUE,...){

  # Argument check on "order"
  # It is not necessary to check on "data" or "target" as they will
  # be handled by makeClassifTask function
  if( !order %in% c(1, 2, 3) | !inherits(order, 'integer')){
    stop('order must be 1, 2, or 3.')
  }

  # create the classification task

  task   <- mlr::makeRegrTask(data = data, target = target, ...)


  if( order %in% c(2, 3)){
    # extract target and descriptive features

    y   <- task$env$data[, target]

    if( order == 2){
      fml <- formula(paste0(target, "~.*."))
    }else{
      fml <- formula(paste0(target, "~.^3"))
    }

    X   <- data.frame(model.matrix(fml, task$env$data))

    # Remove constant terms
    if( remove.constant){
      X    <- mlr::removeConstantFeatures(X)
    }


    # bind the data
    newdata <- cbind(X,y)
    colnames(newdata)[ncol(newdata)] <- target

    # prompt warning message if p > n
    p <- ncol(X)
    if( p > nrow(X)){
      warning('More features than number of observations.')
    }

    # reconfigure the task
    task    <- mlr::makeRegrTask(data = newdata, target = target, ...)
  }

  return(task)

}
```

Automatic Feature Selection Script for Regression Tasks

```
# 0. Preliminary ----
# Load the relevant packages
library(mlr)
library(spFSR)
library(jsonlite)
library(glinternet)

# Source the modificed modified make-regression task with higher orders
source('modifiedMakeRegrTask.R')

# Common learner: linear regression
wrapper     <- makeLearner('regr.lm')

# Datasets and number of CV
data_files  <- c('BostonHousing', 'Ailerons', 'Elevators')
targets     <- c('medv', 'V41', 'V19')
numberCV    <- c(4, 10, 10)

# 1. Learning interactions using SFS for linear regresssion task ----
# Define SFS control
SFSctrl    <- makeFeatSelControlSequential(method = 'sfs',
                                           alpha = 0.001, beta = -0.0005)

for(j in 1:length(data_files)){

  # Initialise the all_result data.frame to store the result
  m <- 0
  results_summary   <- data.frame()

  data_name   <- data_files[j]
  target <- targets[j]

  try(
    {
      data        <- read.csv(paste0(data_files[j], '_train.csv'))
      test_data   <- read.csv(paste0(data_files[j],'_test.csv'))
    }
  )

  task <- makeRegrTask(data = data, target = target)

  # Create a vector of random seeds for reproducibility
  if( data_name == 'BostonHousing'){
    seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
  }else{
    seed_vector <- c(1, 4, 781)
  }


  for(i in 1:length(seed_vector)){

    seed.number <- seed_vector[i]
    set.seed(seed.number)

    m <- m + 1

    # Resampling
    rdesc       <- makeResampleDesc(method = 'CV' , iters = numberCV[j])

    # Extract main effects
    main_sfeats <- selectFeatures(wrapper, task = task, resampling = rdesc,
                                  control = SFSctrl, show.info = TRUE,
                                  measures = mlr::rmse )

    # Obtain the performance with main effects only
    reduced_main_task <- makeRegrTask(id = 'reduced main',
                                      data = data[, c(main_sfeats$x , targets[j])],
                                      target = targets[j])

    # Add interactions
    second_task  <- modifiedMakeRegrTask(id = 'interaction',
                                         data = data[, c(main_sfeats$x , target)],
                                         target = target, order = 2L)

    # Specify the maximum number of trials
    z <- 1
    success <- FALSE

    tryCatch({
      second_sfeats <- selectFeatures(wrapper, task = second_task,
                                      resampling = rdesc,
                                      control = SFSctrl, show.info = TRUE,
                                      measures = mlr::rmse)
      success <- TRUE

    },
    error = function(e){conditionMessage(e)}
    )


    results_summary[m, 'dataset']      <- data_name
    results_summary[m, 'seed_number']  <- seed.number
    results_summary[m, 'p_0']          <- length(main_sfeats$x)

    if(exists('second_sfeats')){
      signif_terms     <- as.character(unique(c(main_sfeats$x , second_sfeats$x)))
      results_summary[m, 'p_1'] <- length(signif_terms) - length(main_sfeats$x)

      # Add the omitted main effects
      third_task <- makeRegrTask(id = 'strong hierarchy',
                                 data =
                                  second_task$env$data[, c(signif_terms, targets[j])],
                                 target = targets[j])

      constrainedMod   <- train( wrapper, third_task )
      constrainedPred  <- predict( constrainedMod, third_task)

      results_summary[m, 'train_rmse'] <- performance( constrainedPred,
                                                       measure = mlr::rmse)
      results_summary[m, 'train_AIC']  <- AIC( constrainedMod$learner.model)
      results_summary[m, 'train_BIC']  <- BIC( constrainedMod$learner.model)
      sub_test_task  <- modifiedMakeRegrTask(data = test_data, target = targets[j],
                                             order = 2L, remove.constant = FALSE)

      sub_test_task <- makeRegrTask(data =
                                   sub_test_task$env$data[, c(signif_terms, targets[j])],
                                  target = targets[j], id = 'test_subset')

      sub_pred_test                  <- predict(constrainedMod , sub_test_task)
      results_summary[m, 'test_rmse']<- performance( sub_pred_test, mlr::rmse)

      output <- list(coeff   = constrainedMod$learner.model$coefficients,
                     result = results_summary[m, ],
                     features = signif_terms)

      write(toJSON(output, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
            paste0('GA_',data_name,'_',seed.number,'.txt'))

    }else{
      results_summary[m, 'p_1']        <- NA
      results_summary[m, 'train_rmse'] <- NA
      results_summary[m, 'train_AIC']  <- NA
      results_summary[m, 'train_BIC']  <- NA
      results_summary[m, 'test_rmse']  <- NA

    }

  }

  write.csv(results_summary, paste0('SFS_regr_',data_files[j],'.csv'),
            row.names = FALSE)

}


# 2. Learning interactions via hierarchical group-lasso regularization ----
for(j in 1:length(data_files)){

  # Initialise the all_result data.frame to store the result
  m <- 0
  data_name  <- data_files[j]
  all_result <- data.frame()

  if( data_name == 'BostonHousing'){
    data  <- read.csv('BostonHousing_train.csv')
    data  <- removeConstantFeatures(data)
    Y     <- data$medv
    X     <- data[, c(1:ncol(data)-1)]

    test_data <- read.csv('BostonHousing_test.csv')
    test_Y    <- test_data$medv
    test_X    <- test_data[, c(1:ncol(test_data)-1)]

  }else if( data_name == 'Ailerons'){

    data  <- read.csv('Ailerons_train.csv')
    data  <- removeConstantFeatures(data)
    Y     <- data$V41
    X     <- data[, c(1:ncol(data)-1)]

    test_data <- read.csv('Ailerons_test.csv')
    test_Y    <- test_data$V41
    test_X    <- test_data[, c(1:ncol(test_data)-1)]


  }else if( data_name == 'Elevators'){

    data  <- read.csv('Elevators_train.csv')
    data  <- removeConstantFeatures(data)
    Y     <- data$V19
    X     <- data[, c(1:ncol(data)-1)]

    test_data <- read.csv('Elevators_test.csv')
    test_Y    <- test_data$V19
    test_X    <- test_data[, c(1:ncol(test_data)-1)]


  }else{
    stop('No dataset found')
  }

  # Create a vector of random seeds for reproducibility
  # Note: Boston Housing has more random seeds as its data size is small
  if( data_name == 'BostonHousing'){
    seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
  }else{
    seed_vector <- c(1, 4, 781)
  }
```

```r
for(i in 1:length(seed_vector)){

  m <- m + 1

  seed.number <- seed_vector[i]
  set.seed( seed.number )

  startTime <- Sys.time()
  fit       <- glinternet.cv(X, Y, numLevels = rep(1, ncol(X)),
                             family = 'gaussian', nFolds = numberCV[j],
                             numCores = 1, verbose = TRUE)
  endTime   <- Sys.time()


  # Obtain main effects and interaction
  coeff <- coef(fit)
  p_0   <- length(coeff$mainEffects$cat) + length(coeff$mainEffects$cont)
  p_1   <- length(coeff$interactions$catcat) +
    length(coeff$interactions$contcat) +
    length(coeff$interactions$contcont)

  k <- p_0 + p_1

  # calculate RMSE
  prediction <- data.frame(truth = Y, pred = fit$fitted)
  train_rmse <- mean((prediction$truth-prediction$pred)^2)^0.5

  # predict on test data
  test_prediction <- data.frame(truth = test_Y, pred = predict(fit, test_X))
  test_rmse <- mean((test_prediction$truth-test_prediction$pred)^2)^0.5

  # Export the granular info as JSON
  result <- list( colnames = colnames(X),
                  mainEffects  = coeff$mainEffects,
                  interactions = coeff$interactions,

                  train_rmse  = train_rmse,
                  test_rmse   = test_rmse,

                  lambda = fit$lambdaHat,
                  runtime = as.numeric(endTime - startTime ))

  write(toJSON(result, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
           paste0('glinternet_', data_name, '_', seed.number, '.txt'))

    # Append the all result summary
    all_result[m, 'dataset']   <- data_files[j]
    all_result[m, 'seed']      <- seed.number
    all_result[m, 'lambda']    <- result$lambda

    all_result[m, 'p_0']       <- p_0
    all_result[m, 'p_1']       <- p_1
    all_result[m, 'train_rmse']<- train_rmse
    all_result[m, 'test_rmse'] <- test_rmse

  }
  # Save the result summary
  write.csv(all_result,
            paste0('glinternet_regr_',data_files[j],'.csv'),
            row.names = FALSE)
}

# 3. Learning interactions using SP-FSR for linear regresssion task ----
for(j in 1:length(data_files)){

  # Initialise the all_result data.frame to store the result
  m <- 0
  results_summary   <- data.frame()

  data_name   <- data_files[j]
  target      <- targets[j]

  try(
    {
      data        <- read.csv(paste0(data_files[j], '_train.csv'))
      test_data   <- read.csv(paste0(data_files[j],'_test.csv'))
    }
  )

  task <- makeRegrTask(data = data, target = target)

  # Create a vector of random seeds for reproducibility
  # Note: Boston Housing has more random seeds as its data size is small
  if( data_name == 'BostonHousing'){
    seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
  }else{
    seed_vector <- c(1, 4, 781)
  }
```

```
for(i in 1:length(seed_vector)){

  seed.number <- seed_vector[i]
  set.seed(seed.number)

  # Auto feature selection of main effects ----
  spsaMod <- spFeatureSelection(  task = task,
                                  wrapper = wrapper,
                                  measure = mlr::rmse,
                                  num.features.selected = 0,
                                  norm.method = NULL,
                                  cv.stratify = FALSE,
                                  num.cv.folds = numberCV[j])

  # Store the result
  m <- m + 1
  results_importance            <- list()
  est_coeff                     <- list()

  results_summary[m, 'dataset']     <- data_name
  results_summary[m, 'seed']        <- seed.number
  results_summary[m, 'p_0']         <- length(spsaMod$features)
  results_summary[m, 'mean_0']      <- spsaMod$best.value
  results_summary[m, 'std_0']       <- spsaMod$best.std
  results_summary[m, 'runtime_0']   <- spsaMod$run.time

  k <- 1
  results_importance[[k]]   <- getImportance(spsaMod)
  features.to.keep <- as.character(results_importance[[k]]$features)
  target           <- task$task.desc$target

  fittedTask     <- makeRegrTask(data[, c(features.to.keep, target)],
                                 target = target, id = 'subset')

  fittedMod      <- train(wrapper, fittedTask)
  pred           <- predict(fittedMod, fittedTask)
  fittedMod      <- fittedMod$learner.model
  est_coeff[[k]] <- data.frame(coefficient = fittedMod$coefficients)


  # Select interactions ----
  sub_task  <- modifiedMakeRegrTask(data = data[, c(features.to.keep, target)],
                                    target = target, order = 2L)
```

```
sub_spsaMod <- spFeatureSelection(  task = sub_task,
                                    wrapper = wrapper,
                                    measure = mlr::rmse,
                                    num.features.selected = 0,
                                    norm.method = NULL,
                                    features.to.keep = features.to.keep,
                                    cv.stratify = FALSE,
                                    num.cv.folds = numberCV[j])

k <- k + 1
results_summary[m, 'p_1']       <- length(sub_spsaMod$features)
results_summary[m, 'mean_1']    <- sub_spsaMod$best.value
results_summary[m, 'std_1']     <- sub_spsaMod$best.std
results_summary[m, 'runtime_1'] <- sub_spsaMod$run.time
results_importance[[k]]         <- getImportance(sub_spsaMod)

new_features   <- as.character(results_importance[[k]]$features)
sub_fittedtask <- makeRegrTask(sub_task$env$data[, c(new_features, target)],
                              target = target, id = 'subset')

sub_fittedMod  <- train(wrapper, sub_fittedtask)
sub_pred       <- predict(sub_fittedMod, sub_fittedtask)

est_coeff[[k]]  <- data.frame(coefficient = sub_fittedMod$learner.model$coefficients)

results_summary[m, 'train_AIC']     <- AIC( sub_fittedMod$learner.model)
results_summary[m, 'train_BIC']     <- BIC( sub_fittedMod$learner.model )
results_summary[m, 'train_rmse']    <- mlr::performance(sub_pred, mlr::rmse)



# Predict on the test data
sub_test_task <- modifiedMakeRegrTask(data =
                                        test_data[, c(features.to.keep, target)],
                                      target = target, order = 2L,
                                      remove.constant = FALSE)

sub_test_task <- makeRegrTask(data =
                                sub_test_task$env$data[, c(new_features, target)],
                              target = target, id = 'test_subset')

sub_pred_test  <- predict(sub_fittedMod, sub_test_task)

results_summary[m, 'test_rmse']     <- mlr::performance( sub_pred_test, mlr::rmse)
```

```
    # Prediction w/o interaction terms
    test_task        <- makeRegrTask(data = test_data, target = target)
    test_pred_noint  <- predict(spsaMod$best.model, test_task)
    train_pred_noint <- predict(spsaMod$best.model, task)

    results_summary[m, 'test_rmse']       <- mlr::performance( sub_pred_test, mlr::rmse)
    results_summary[m, 'test_rmse_noint'] <- mlr::performance(test_pred_noint, mlr::rmse)
    results_summary[m, 'train_rmse_noint']<- mlr::performance(train_pred_noint, mlr::rmse)

    final_result   <- list(features = new_features,
                           est_coeff = est_coeff,
                           results_importance,
                           result = results_summary[m, ])

    write(toJSON(final_result, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('spfsr_',data_name,'_',seed.number,'.txt'))
  }

  write.csv(results_summary,
            paste0('spfsr_regr_',data_files[j],'.csv'),
            row.names = FALSE)

}
# 4. Learning interactions using GA for linear regresssion task ----
# Define GA control
GActrl <-  makeFeatSelControlGA(maxit = 100)

for(j in 1:length(data_files)){
  # Initialise the all_result data.frame to store the result
  m <- 0
  results_summary   <- data.frame()
  data_name   <- data_files[j]

  try(
    {
      data        <- read.csv(paste0(data_files[j], '_train.csv'))
      test_data   <- read.csv(paste0(data_files[j],'_test.csv'))
    }
  )

  task <- makeRegrTask(data = data, target = targets[j])

  # Create a vector of random seeds for reproducibility
  # Note: Boston Housing has more random seeds as its data size is small
```

```r
if( data_name == 'BostonHousing'){
  seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
}else{
  seed_vector <- c(1, 4, 781)
}

for(i in 1:length(seed_vector)){

  seed.number <- seed_vector[i]
  set.seed(seed.number)

  m <- m + 1

  # Resampling
  rdesc       <- makeResampleDesc(method = 'CV' , iters = numberCV[j])

  # Extract main effects
  main_sfeats <- selectFeatures(wrapper, task = task, resampling = rdesc,
                                control = GActrl, show.info = TRUE,
                                measures = mlr::rmse )

  # Obtain the performance with main effects only
  reduced_main_task <- makeRegrTask(id = 'reduced main',
                                    data = data[, c(main_sfeats$x , targets[j])],
                                    target = targets[j])

  # Add interactions
  second_task  <- modifiedMakeRegrTask(id = 'interaction',
                                       data = data[, c(main_sfeats$x , targets[j])],
                                       target = targets[j], order = 2L)

  second_sfeats <- selectFeatures(wrapper, task = second_task,
                                  resampling = rdesc,
                                  control = GActrl, show.info = TRUE,
                                  measures = mlr::rmse )

  results_summary[m, 'dataset']      <- data_name
  results_summary[m, 'seed_number']  <- seed.number
  results_summary[m, 'p_0']          <- length(main_sfeats$x)

  signif_terms     <- as.character(unique(c(main_sfeats$x , second_sfeats$x)))
  results_summary[m, 'p_1'] <- length(signif_terms) - length(main_sfeats$x)

  # Add the omitted main effects
  third_task <- makeRegrTask(id = 'strong hierarchy',
                              data = second_task$env$data[, c(signif_terms, targets[j])],
                              target = targets[j])

    constrainedMod   <- train( wrapper, third_task )
    constrainedPred  <- predict( constrainedMod, third_task)


    results_summary[m, 'train_rmse'] <- performance( constrainedPred , measure = mlr::rmse)
    results_summary[m, 'train_AIC'] <- AIC( constrainedMod$learner.model)
    results_summary[m, 'train_BIC'] <- BIC( constrainedMod$learner.model)

    sub_test_task  <- modifiedMakeRegrTask(data = test_data, target = targets[j],
                                          order = 2L, remove.constant = FALSE)

    sub_test_task  <- makeRegrTask(sub_test_task$env$data[, c(signif_terms, targets[j])],
                                  target = targets[j], id = 'test_subset')

    sub_pred_test         <- predict(constrainedMod , sub_test_task)
    results_summary[m, 'test_rmse'] <- performance( sub_pred_test, measure = mlr::rmse)


    output <- list(coeff  = constrainedMod$learner.model$coefficients,
                   result = results_summary[m, ],
                   features = signif_terms)

    write(toJSON(output, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('GA_',data_name,'_',seed.number,'.txt'))

  }

  write.csv(results_summary,
            paste0('GA_regr_',data_files[j],'.csv'),
            row.names = FALSE)

}
```

R Codes for Classification Tasks

Baseline Learner for Classification

```r
# Baseline learners without interactions: classification tasks ----
# Compare baseline learners with other methods
# 0, Load packages ----
library(mlr)
library(tidyverse)
```

```r
# 1. Run baseline learners ----
# Initialise the data frame to store results,
# Prepare data sets and target continuous features
result      <- data.frame()

data_files  <- c('Sonar', 'ionosphere')
targets     <- c('Class', 'class')

# Common learner: logistic regression
wrapper <- makeLearner('classif.logreg', predict.type = 'prob')

# 2. Loop for each dataset ----
for(j in 1:length(data_files) ){

  train_data  <- read.csv(paste0(data_files[j], '_train.csv'))
  test_data   <- read.csv(paste0(data_files[j],'_test.csv'))

  train_task  <- makeClassifTask(data = train_data, target = targets[j])
  test_task   <- makeClassifTask(data = test_data, target = targets[j])

  baseModel   <- train(wrapper, train_task)
  train_pred  <- predict(baseModel, train_task)
  test_pred   <- predict(baseModel, test_task)

  result[j, 'Data'] <- data_files[j]
  result[j, 'AIC']  <- AIC(baseModel$learner.model)
  result[j, 'BIC']  <- BIC(baseModel$learner.model)
  result[j, 'Train_logLoss'] <- mlr::performance(train_pred, mlr::logloss)
  result[j, 'Test_logLoss']  <- mlr::performance(test_pred, mlr::logloss)
  result[j, 'Train_auc']     <- mlr::performance(train_pred, mlr::auc)
  result[j, 'Test_auc']      <- mlr::performance(test_pred, mlr::auc)
}


# Save the result as a csv file
write.csv(result, 'baselearner_classif.csv', row.names = FALSE)
```

Modified Make Classification Task Function

```r
#' @description Generating pairwise interaction terms of a data
#'              and pass in to a task
#' @param data A data frame containing the features and target variable(s)
#' @param target Name of the target variable
#' @param order Interaction order. The allowed values are 1, 2, and 3.
#'             Default is 2L
```

```
modifiedMakeClassifTask <- function(data, target, order = 2,
                                    remove.constant = FALSE,...){

  # Argument check on "order"
  # It is not necessary to check on "data" or "target" as they will
  # be handled by makeClassifTask function
  if( !order %in% c(1, 2, 3) | !inherits(order, 'integer')){
    stop('order must be 1, 2, or 3.')
  }

  # create the classification task

  task   <- mlr::makeClassifTask(data = data, target = target, ...)

  if( order %in% c(2, 3)){
    # extract target and descriptive features

    y   <- task$env$data[, target]

    if( order == 2){
      fml <- formula(paste0(target, "~.*."))
    }else{
      fml <- formula(paste0(target, "~.^3"))
    }

    X   <- data.frame(model.matrix(fml, task$env$data))

    # Remove constant terms
    if(remove.constant){
      X   <- mlr::removeConstantFeatures(X)
    }

    # bind the data
    newdata <- cbind(X,y)
    colnames(newdata)[ncol(newdata)] <- target

    # prompt warning message if p > n
    p <- ncol(X)
    if( p > nrow(X)){
      warning('More features than number of observations.')
    }

    # reconfigure the task
    task   <- mlr::makeClassifTask(data = newdata, target = target, ...)
  }

  return(task)

}
```

Automatic Feature Selection Script for Classification Tasks

```
# 0. Preliminary ----
# Load the relevant packages
library(mlr)
library(spFSR)
library(jsonlite)
library(glinternet)
library(pROC)
library(MASS)
library(ROCR)

source('modifiedMakeClassifTask.R')

# Create a vector of random seeds for reproducibilty
seed_vector <- c(1, 1010, 4, 781, 9990, 9999, 81, 46, 67, 2, 3, 1017)
measure     <- mlr::auc

# Configuration of common task, wrapper and control
wrapper  <- makeLearner('classif.logreg', predict.type = 'prob')
rdesc    <- makeResampleDesc(method = 'CV', stratify = TRUE, iters = 5)

# 1. Learning interactions with SFFS ----
ctrl     <- makeFeatSelControlSequential(method = 'sffs',
                                         alpha = 0.001,
                                         beta = 0.0005)


m <- 0
result   <- data.frame()
datasets <- c('ionosphere', 'Sonar')

for( j in 1:length(datasets)){

  data_name <- datasets[j]
  if( data_name == 'ionosphere'){

    data      <- read.csv('ionosphere_train.csv')
    target    <- 'class'
    test_data <- read.csv('ionosphere_test.csv')
```

```r
}else if( data_name == 'Sonar'){

  data      <- read.csv('Sonar_train.csv')
  target    <- 'Class'
  test_data <- read.csv('Sonar_test.csv')

}else{
  stop('No dataset found')
}

data  <- removeConstantFeatures(data)
data  <- na.omit(data)

test_data <- removeConstantFeatures(test_data)
task      <- makeClassifTask(data = data, target = target)

for(i in 1:length(seed_vector)){

  seed.number <- seed_vector[i]
  set.seed(seed.number)

  m <- m + 1

  # Extract main effects
  main_sfeats   <- selectFeatures(wrapper, task = task, resampling = rdesc,
                                  control = ctrl, show.info = TRUE, measures = measure )


  # Obtain the performance with main effects only
  reduced_main_task <- makeClassifTask(id = 'reduced main',
                                       data = data[, c(main_sfeats$x , target)],
                                       target = target)


  # Add interactions
  second_task <- modifiedMakeClassifTask(id = 'interaction',
                                         data = data[, c(main_sfeats$x , target)],
                                         target = target, order = 2L)

  second_sfeats <- selectFeatures(wrapper, task = second_task, resampling = rdesc,
                                  control = ctrl, show.info = TRUE, measures = measure )

  result[m, 'dataset']      <- data_name
  result[m, 'seed_number']  <- seed.number
  result[m, 'p_0']          <- length(main_sfeats$x)
```

```r
    signif_terms     <- as.character(unique(c(main_sfeats$x , second_sfeats$x)))
    result[m, 'p_1'] <- length(signif_terms) - length(main_sfeats$x)

    # Add the omitted main effects
    third_task <- makeClassifTask( id = 'strong hierarchy',
                                   data = second_task$env$data[, c(signif_terms, target)],
                                   target = target)

    constrainedMod   <- train( wrapper, third_task )
    constrainedPred  <- predict( constrainedMod, third_task)


    result[m, 'train_auc'] <- performance( constrainedPred , measure = measure)
    result[m, 'train_AIC'] <- AIC( constrainedMod$learner.model)
    result[m, 'train_BIC'] <- BIC( constrainedMod$learner.model)

    sub_test_task  <- modifiedMakeClassifTask(data = test_data, target = target, order = 2L)

    sub_test_task  <- makeClassifTask(sub_test_task$env$data[, c(signif_terms, target)],
                                      target = target, id = 'test_subset')

    sub_pred_test          <- predict(constrainedMod , sub_test_task)
    result[m, 'test_auc'] <- performance( sub_pred_test, measure)

    output <- list(coeff  = constrainedMod$learner.model$coefficients,
                   result = result[m, ],
                   features = signif_terms)

    write(toJSON(output, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('SFFS_',data_name,'_',seed.number,'.txt'))
  }

}

write.csv(result, 'SFFS_results.csv', row.names = FALSE)



# 2. Learning interactions with two-step SP-FSR ----
# Reset the results summary
m <- 0
results_summary   <- data.frame()

for(j in 1:length(datasets)){
```

```r
data_name  <- datasets[j]

if( data_name == 'ionosphere'){

  data      <- read.csv('ionosphere_train.csv')
  target    <- 'class'
  test_data <- read.csv('ionosphere_test.csv')
  data      <- removeConstantFeatures(data)
  test_data <- removeConstantFeatures(test_data)

}else if( data_name == 'Sonar'){

  data      <- read.csv('Sonar_train.csv')
  target    <- 'Class'
  test_data <- read.csv('Sonar_test.csv')
  data      <- removeConstantFeatures(data)
  test_data <- removeConstantFeatures(test_data)

}else{
  stop('No dataset found')
}

task      <- makeClassifTask(data = data, target = target)


for(i in 1:length(seed_vector)){

  seed.number <- seed_vector[i]
  set.seed(seed.number)

  # Auto feature selection of main effects ----
  spsaMod <- spFeatureSelection(  task = task,
                                  wrapper = wrapper,
                                  measure = measure,
                                  num.features.selected = 0,
                                  norm.method = NULL)

  # Store the result
  m <- m + 1
  results_importance             <- list()
  est_coeff                      <- list()

  results_summary[m, 'dataset']      <- data_name
  results_summary[m, 'seed']         <- seed.number
```

```r
    results_summary[m, 'p_0']         <- length(spsaMod$features)
    results_summary[m, 'mean_0']      <- spsaMod$best.value
    results_summary[m, 'std_0']       <- spsaMod$best.std
    results_summary[m, 'runtime_0']   <- spsaMod$run.time

    k <- 1
    results_importance[[k]]   <- getImportance(spsaMod)
    features.to.keep <- as.character(results_importance[[k]]$features)
    target           <- task$task.desc$target

    fittedTask     <- makeClassifTask(data[, c(features.to.keep, target)],
                                      target = target, id = 'subset')

    fittedMod      <- train(wrapper, fittedTask)
    pred           <- predict(fittedMod, fittedTask)
    fittedMod      <- fittedMod$learner.model
    est_coeff[[k]] <- data.frame(coefficient = fittedMod$coefficients)


    # Select interactions ----
    sub_task  <- modifiedMakeClassifTask(data = data[, c(features.to.keep, target)],
                                         target = target, order = 2L)

    sub_spsaMod <- spFeatureSelection(  task = sub_task,
                                        wrapper = wrapper,
                                        measure = measure,
                                        num.features.selected = 0,
                                        norm.method = NULL,
                                        features.to.keep = features.to.keep)

    k <- k + 1
    results_summary[m, 'p_1']       <- length(sub_spsaMod$features)
    results_summary[m, 'mean_1']    <- sub_spsaMod$best.value
    results_summary[m, 'std_1']     <- sub_spsaMod$best.std
    results_summary[m, 'runtime_1'] <- sub_spsaMod$run.time
    results_importance[[k]]         <- getImportance(sub_spsaMod)

    new_features   <- as.character(results_importance[[k]]$features)
    sub_fittedtask <- makeClassifTask(sub_task$env$data[, c(new_features, target)],
                                      target = target, id = 'subset')
    sub_fittedMod  <- train(wrapper, sub_fittedtask)
    sub_pred       <- predict(sub_fittedMod, sub_fittedtask)

    est_coeff[[k]]  <- data.frame(coefficient = sub_fittedMod$learner.model$coefficients)
```

```
  results_summary[m, 'train_AIC']     <- AIC( sub_fittedMod$learner.model)
  results_summary[m, 'train_BIC']     <- BIC( sub_fittedMod$learner.model )
  results_summary[m, 'train_auc']     <- mlr::performance(sub_pred, mlr::auc)
  results_summary[m, 'train_logloss'] <- mlr::performance(sub_pred, mlr::logloss)


  # Predict on the test data
  sub_test_task  <- modifiedMakeClassifTask(data =
                                             test_data[, c(features.to.keep, target)],
                                           target = target, order = 2L)

  sub_test_task  <- makeClassifTask(sub_test_task$env$data[, c(new_features, target)],
                                   target = target, id = 'test_subset')

  sub_pred_test  <- predict(sub_fittedMod, sub_test_task)

  results_summary[m, 'test_auc']     <- mlr::performance( sub_pred_test, measure)
  results_summary[m, 'test_logloss'] <- mlr::performance( sub_pred_test, mlr::logloss)

  # Prediction w/o interaction terms
  test_task        <- makeClassifTask(data = test_data, target = target)
  test_pred_noint  <- predict(spsaMod$best.model, test_task)
  train_pred_noint <- predict(spsaMod$best.model, task)

  results_summary[m, 'test_auc']     <- mlr::performance( sub_pred_test, measure)
  results_summary[m, 'test_logloss'] <- mlr::performance( sub_pred_test, mlr::logloss)

  results_summary[m, 'test_auc_noint']     <- mlr::performance(test_pred_noint,
                                                              mlr::auc)
  results_summary[m, 'test_logloss_noint'] <- mlr::performance(test_pred_noint,
                                                              mlr::logloss)

  results_summary[m, 'train_auc_noint']     <- mlr::performance(train_pred_noint,
                                                               mlr::auc)
  results_summary[m, 'train_logloss_noint'] <- mlr::performance(train_pred_noint,
                                                               mlr::logloss)

  final_result   <- list(features = new_features,
                         est_coeff = est_coeff,
                         results_importance,
                         result = results_summary[m, ])

  write(toJSON(final_result, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
        paste0('spfsr_',data_name,'_',seed.number,'.txt'))
}
```

```
}

write.csv(results_summary, 'spfsr_results.csv', row.names = FALSE)


# 3. Learning interactions with glinternet ----
# Reset the data frame
m <- 0
all_result <- data.frame()

for(j in 1:length(datasets)){

  data_name  <- datasets[j]

  if( data_name == 'ionosphere'){

    data <- read.csv('ionosphere_train.csv')
    Y    <- ifelse(data$class == 'b', 1, 0)
    data  <- removeConstantFeatures(data)
    X     <- data[, c(1:ncol(data)-1)]

    test_data <- read.csv('ionosphere_test.csv')
    test_data <- removeConstantFeatures(test_data)
    test_Y    <- ifelse(test_data$class == 'b', 1, 0)
    test_X    <- test_data[, c(1:ncol(test_data)-1)]

  }else if( data_name == 'Sonar'){

    data <- read.csv('Sonar_train.csv')
    Y    <- ifelse(data$Class == 'R', 1, 0)
    data  <- removeConstantFeatures(data)
    X     <- data[, c(1:ncol(data)-1)]

    test_data <- read.csv('Sonar_test.csv')
    test_data <- removeConstantFeatures(test_data)
    test_Y    <- ifelse(test_data$Class == 'R', 1, 0)
    test_X    <- test_data[, c(1:ncol(test_data)-1)]

  }else{
    stop('No dataset found')
  }

  for(i in 1:length(seed_vector)){

    m <- m + 1
    seed.number <- seed_vector[i]
    set.seed( seed.number )


    # hierarchical group-lasso regularization (CV = 10)
    startTime <- Sys.time()
    fit       <- glinternet.cv(X, Y, numLevels = rep(1, ncol(X)),
                               family = 'binomial', nFolds = 5)
    endTime   <- Sys.time()

    # Obtain main effects and interaction
    coeff <- coef(fit)
    p_0 <- length(coeff$mainEffects$cat) +
      length(coeff$mainEffects$cont)
    p_1 <- length(coeff$interactions$catcat) +
      length(coeff$interactions$contcat) +
      length(coeff$interactions$contcont)

    k <- p_0 + p_1

    # calculate AIC and BIC
    prediction <- data.frame(truth = Y, pred = fit$fitted)
    logLik     <- sum(prediction$truth*log(prediction$pred) + (1-prediction$truth)*log(1-pre
    AIC        <- -2*logLik + 2*k
    BIC        <- -2*logLik + log(nrow(X))*k

    # predict on test data
    test_prediction <- data.frame(truth = test_Y,
                                  pred = predict(fit, test_X))

    test_logLik    <- sum(test_prediction$truth*log(test_prediction$pred) +
                          (1-test_prediction$truth)*log(1-test_prediction$pred))

    test_AIC        <- -2*test_logLik + 2*k
    test_BIC        <- -2*test_logLik + log(nrow(test_X))*k


    # Store the result
    result <- list( colnames = colnames(X),
                    mainEffects  = coeff$mainEffects,
                    interactions = coeff$interactions,

                    train_auc  = auc(Y, fit$fitted),
                    AIC = AIC,
```

```r
                    BIC = BIC,

                    test_auc = auc(test_prediction$truth, test_prediction$pred),
                    test_AIC = test_AIC,
                    test_BIC = test_BIC,


                    train_logloss = -logLik/nrow(data),
                    test_logloss  = -test_logLik/nrow(test_data),
                    lambda = fit$lambdaHat,
                    runtime = as.numeric(endTime - startTime ))

    write(toJSON(result, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('glinternet_', data_name, '_', seed.number, '.txt'))

    # Store the all result summary

    all_result[m, 'dataset']   <- datasets[j]
    all_result[m, 'seed']      <- seed.number
    all_result[m, 'lambda']    <- result$lambda
    all_result[m, 'train_auc'] <- result$train_auc
    all_result[m, 'train_AIC'] <- result$AIC
    all_result[m, 'train_BIC'] <- result$BIC
    all_result[m, 'train_logloss'] <- result$train_logloss
    all_result[m, 'p_0']       <- p_0
    all_result[m, 'p_1']       <- p_1
    all_result[m, 'test_auc']  <- result$test_auc
    all_result[m, 'test_AIC']  <- result$test_AIC
    all_result[m, 'test_BIC']  <- result$test_BIC
    all_result[m, 'test_logloss'] <- result$test_logloss

  }

}

# Save the result summary
write.csv(all_result, 'glinternet_all_result.csv', row.names = FALSE)

# 4. Learning interactions with GA ----
m         <- 0
result    <- data.frame()
ctrl      <- makeFeatSelControlGA( maxit = 20)

for( j in 1:length(datasets)){
```

```r
data_name <- datasets[j]
if( data_name == 'ionosphere'){

  data      <- read.csv('ionosphere_train.csv')
  target    <- 'class'
  test_data <- read.csv('ionosphere_test.csv')

}else if( data_name == 'Sonar'){

  data      <- read.csv('Sonar_train.csv')
  target    <- 'Class'
  test_data <- read.csv('Sonar_test.csv')

}else{
  stop('No dataset found')
}

data  <- removeConstantFeatures(data)
data  <- na.omit(data)

test_data <- removeConstantFeatures(test_data)
task      <- makeClassifTask(data = data, target = target)

for(i in 1:length(seed_vector)){

  seed.number <- seed_vector[i]
  set.seed(seed.number)

  m <- m + 1

  # Extract main effects
  main_sfeats   <- selectFeatures(wrapper, task = task, resampling = rdesc,
                                  control = ctrl, show.info = TRUE, measures = measure )


  # Obtain the performance with main effects only
  reduced_main_task <- makeClassifTask(id = 'reduced main',
                                       data = data[, c(main_sfeats$x , target)],
                                       target = target)


  # Add interactions
  second_task <- modifiedMakeClassifTask(id = 'interaction',
                                         data = data[, c(main_sfeats$x , target)],
                                         target = target, order = 2L)
```

```
    second_sfeats <- selectFeatures(wrapper, task = second_task, resampling = rdesc,
                                  control = ctrl, show.info = TRUE, measures = measure )

    result[m, 'dataset']      <- data_name
    result[m, 'seed_number']  <- seed.number
    result[m, 'p_0']          <- length(main_sfeats$x)

    signif_terms     <- as.character(unique(c(main_sfeats$x , second_sfeats$x)))
    result[m, 'p_1'] <- length(signif_terms) - length(main_sfeats$x)

    # Add the omitted main effects
    third_task <- makeClassifTask( id = 'strong hierarchy',
                                 data = second_task$env$data[, c(signif_terms, target)],
                                 target = target)

    constrainedMod   <- train( wrapper, third_task )
    constrainedPred  <- predict( constrainedMod, third_task)


    result[m, 'train_auc'] <- performance( constrainedPred , measure = measure)
    result[m, 'train_AIC'] <- AIC( constrainedMod$learner.model)
    result[m, 'train_BIC'] <- BIC( constrainedMod$learner.model)

    sub_test_task  <- modifiedMakeClassifTask(data = test_data, target = target, order = 2L)

    sub_test_task  <- makeClassifTask(sub_test_task$env$data[, c(signif_terms, target)],
                                    target = target, id = 'test_subset')

    sub_pred_test         <- predict(constrainedMod , sub_test_task)
    result[m, 'test_auc'] <- performance( sub_pred_test, measure)

    output <- list(coeff  = constrainedMod$learner.model$coefficients,
                 result = result[m, ],
                 features = signif_terms)

    write(toJSON(output, auto_unbox = TRUE, pretty = TRUE, factor = 'string'),
          paste0('GA_',data_name,'_',seed.number,'.txt'))
  }

}

write.csv(result, 'GA_result.csv', row.names = FALSE)
```

References

Aksakalli, Vural, Babak Abbasi, and Yong Kai Wong. 2018. *SpFSR: Feature Selection and Ranking by Simultaneous Perturbation Stochastic Approximation.*

Aksakalli, Vural, and Milad Malekipirbazari. 2016. "Feature Selection via Binary Simultaneous Perturbation Stochastic Approximation." *Pattern Recognition Letters* 75 (Supplement C): 41–47. doi:https://doi.org/10.1016/j.patrec.2016.03.002.

Barzilai, J., and J. Borwein. 1988. "Two-Point Step Size Gradient Methods." *IMA Journal of Numerical Analysis* 8: 141–48.

Bien, Jacob, Jonathan Taylor, and Robert Tibshirani. 2013. "A Lasso for Hierarchical Interactions." *Ann. Statist.* 41 (3). The Institute of Mathematical Statistics: 1111–41. doi:10.1214/13-AOS1096.

Bilder, Thomas M., Christopher R.and Loughin. 2015. "Analysis of Categorical Data with R." CRC Press.

Bischl, Bernd, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. 2016. "mlr: Machine Learning in R." *Journal of Machine Learning Research* 17 (170): 1–5. http://jmlr.org/papers/v17/15-066.html.

Cox, D. R. 1984. "Interaction." *International Statistical Review / Revue Internationale de Statistique* 52 (1): 1–24. http://www.jstor.org/stable/1403235.

Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software, Articles* 33 (1): 1–22. doi:10.18637/jss.v033.i01.

Hamilton, Lawrence C. 2011. "Education, Politics and Opinions About Climate Change Evidence for Interaction Effects." *Climatic Change* 104 (2): 231–42. doi:10.1007/s10584-010-9957-8.

Kohavi, R., and G. H. John. 1997. "Wrappers for Feature Subset Selection." *Artificial Intelligence* 97 (1-2): 273–324.

Li, Yang, and Jun S. Liu. 2017. "Robust Variable and Interaction Selection for Logistic Regression and General Index Models." *Journal of the American Statistical Association* 0 (ja). Taylor & Francis: 0–0. doi:10.1080/01621459.2017.1401541.

Lichman, M. 2013. "UCI Machine Learning Repository." University of California, Irvine, School of Information; Computer Sciences. http://archive.ics.uci.edu/ml.

Lim, Michael, and Trevor Hastie. 2018. *Glinternet: Learning Interactions via Hierarchical Group-Lasso Regularization.* https://CRAN.R-project.org/package=glinternet.

Peng, H., F. Long, and C. Ding. 2005. "Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 1226–1238.

Pudil, P., F. J. Ferri, J. Novovicova, and J. Kittler. 1994. "Floating Search Methods for

Feature Selection with Nonmonotonic Criterion Functions.” In *Proceedings of the 12th Iapr International Conference on Pattern Recognition, Vol. 3 - Conference c: Signal Processing (Cat. No.94CH3440-5)*, 2:279–83 vol.2. doi:10.1109/ICPR.1994.576920.

Schwender, Holger, and Katja Ickstadt. 2008. “Identification of Snp Interactions Using Logic Regression.” *Biostatistics* 9 (1): 187–98. doi:10.1093/biostatistics/kxm024.

Shah, Rajen D. 2016. “Modelling Interactions in High-Dimensional Data with Backtracking.” *J. Mach. Learn. Res.* 17 (1). JMLR.org: 7225–55. http://dl.acm.org/citation.cfm?id=2946645.3053489.

Siedlecki, W., and J. Sklansky. 2011. “A Note on Genetic Algorithm for Large-Scale Feature Selection.” In *Handbook of Pattern Recognition and Computer Vision*, 88–107. doi:10.1142/9789814343138_0005.

Sikonia, M. R., and I. Kononenko. 2003. “Theoretical and Empirical Analysis of Relief and Relieff.” *Machine Learning* 53 (23-69).

Simon, Noah Simon, and Robert Tibshirani. 2012. “A Permutation Approach to Testing Interactions in Many Dimensions,” June. https://arxiv.org/abs/1206.6519.

Spall, James C. 1992. “Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation.” *IEEE* 37 (3): 322–41.

Spall, James C. 2003. *’Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control’*. John Wiley.

Spall, James C., and Wang Qi. 2011. “Discrete Simultaneous Perturbation Stochastic Approximation on Loss Function with Noisy Measurements.” *’In: Proceeding American Control Conference’* 37 (3): 4520–5.

Tibshirani, Robert. 1996. “Regression Shrinkage and Selection via the Lasso.” *Journal of the Royal Statistical Society, Series B* 58: 267–88.

Torgo, L. 2017. “DCC Regression Datasets.” Universidade Do Porto, Portugal, Department of Computer Science. http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html.

Yenice, Zeren D., Niranjan Adhikari, Yong Kai Wong, Vural Aksakalli, Alev Taskin Gumus, and Babak Abbasi. 2018. “SPSA-Fsr: Simultaneous Perturbation Stochastic Approximation in Feature Selection and Ranking.” https://arxiv.org/abs/1804.05589.

Yuan, Ming, and Yi Lin. 2006. “Model Selection and Estimation in Regression with Grouped Variables” 68 (February): 49–67.