High Dimensional Selection with Interactions Algorithm on Feature Selection for Binary Outcome

by
Ziqian Zhuang
supervised by
Wei Xu
Rahi Jain

Toronto, Ontario, Canada
April 26, 2021

Contents

Abstract

Feature selection on high-dimensional data with interaction effects is a challenging problem. Although most commonly used algorithms, for example, lasso-typed algorithms, can handle high dimensional data, they do not consider interaction effects. To overcome this drawback, a high dimensional Selection with Interactions (HDSI) algorithm was proposed to incorporate interaction terms and existing techniques can be combined with HDSI to do high dimensional data feature selection. The purpose of our work is to extend the HDSI algorithm to binary outcome data (HDSI-BO) as the HDSI algorithm is only applied and assessed in continuous outcome data.

To modify the HDSI algorithm to fit for binary outcome data feature selection, proper performance measurements and feature selection criteria have been specially designed. Simulation and real data studies were carried out to assess the performance of the HDSI-BO algorithm. In the studies, the algorithm has been combined with standard logistic regression, lasso, ridge regression, elastic net, and adaptive lasso to verify its applicability. The feature selection and predictive performance of these HDSI-BO methods were compared with the standard methods.

Results of simulation and real data studies confirm the applicability of the HDSI-BO algorithm. Furthermore, the HDSI-BO methods have a better ability to select true marginal features and interaction terms and consequently, have the predictive ability with higher levels of accuracy.

keywords: Feature selection; High dimensional data; Interaction terms; HDSI.

1 Introduction

Feature selection is fundamental to high dimensional data problems, for improving the efficiency of data modelings and the predictive ability of models. The growth of data both in sample size (n) and feature dimension (p) has posed various challenges. There could be a large number of features in the original data that are not related to the target research questions. Removing these features, commonly known as noisy features, by feature selection approaches before fitting final models can efficiently improve the precision of estimation on feature effects, measured by model coefficients, and the model prediction accuracy.

Many feature selection strategies have been proposed. The most primary way is to rely on the domain knowledge or experience of experts to shortlist significant variables(Heinze, Wallisch and Dunkler, 2018). But it's very subjective and it will be a big challenge for experts if the feature dimension is high. Subset selection approaches based on standard linear regression only work when $p < n$. Penalized regression approaches such as lasso, ridge, etc. can achieve feature selection regardless of data dimensions through their own penalty principles or artificially predefined threshold values. However, they have their limitations. lasso has built-in feature selection but cannot select features more than n (Emmert-Streib and Dehmer, 2019). When there are highly correlated feature groups, lasso tends to randomly select only one feature and shrink the coefficients of other correlated features to 0 (Zou and

Hastie, 2005). Ridge regression can solve the multicollinearity problem by shrinking some coefficient estimates close to 0 but not exactly 0, but doing so means ridge regression loses the ability to do feature selection. Elastic net break through the limitation of lasso and ridge regression. It supports group selection and can select more than n features (Zou and Hastie, 2005). Nonetheless, it inevitably selects some noise variables. As an extension of the above lasso-type approaches, adaptive lasso has nicer properties. However, its performance heavily depends on the accuracy of the OLS estimates of weights (Kang and Guo, 2009). As a result, adaptive lasso may suffer from multicollinearity problems, which lead to estimation bias.

Moreover, marginal features might not contain the complete information we need. Hence taking interaction terms into account to fit models sometimes becomes necessary. However, traditionally, up until the development of model fitting algorithms, most algorithms do not consider interaction terms directly despite interaction terms being added to models as necessary(Hahn, Ritchie and Moore, 2003).

To circumvent the problems mentioned above as well as improve the accuracy of feature selection, researchers proposed a new algorithm, named High Dimensional Selection with Interactions (HDSI) algorithm for feature selection (Jain and Xu, 2021). HDSI is a flexible framework that can be combined with multiple existing statistical techniques as well as incorporate interaction terms to do feature selection for high dimensional data. HDSI has been combined with several standard techniques, including simple linear

regression, lasso, and adaptive lasso, and applied to data with a continuous outcome feature. Simulation studies and real data studies have been carried out to compare the performance of HDSI and these standard techniques. In general, HDSI outperforms the commonly used standard algorithms in the ability to select true features and the prediction precision.

To expand the application of HDSI, we propose the HDSI-BO algorithm based on HDSI to handle binary outcome data. We design proper performance measurements and feature selection principles to fit for binary outcome data. Further, we combine HDSI-BO with multiple techniques such as logistic regression, lasso, ridge regression, elastic net and adaptive lasso to prove the wide applicability of the algorithm.

Section section 2 introduces the algorithm of HDSI and the performance measurements and feature selection criteria used in HDSI for binary outcome data. In section section 3, we introduce the simulation studies and real data studies that compare the performance of HDSI with multiple standard methods. and the results of the simulation studies are presented and interpreted. Finally, we conclude the research with a discussion in section 4.

2 Methods

2.1 HDSI-BO Algorithm

HDSI-BO algorithm, developed based on HDSI algorithm, inherits the idea of incorporating interaction terms and flexibility in combination with multiple algorithms. Furthermore, different performance measurements and feature selection criteria are specially designed for binary outcome data. The algorithm is summarized in Figure 1.

For original data with sample size n and feature dimension p, suppose ω-level interaction terms are considered, the algorithm of HDSI-BO is as follows:

(1) **Bootstrapping**: bootstrap B samples with size n from the original dataset. For each bootstrap sample, randomly select $q(q < p)$ features from the original feature space. Now the term set for one bootstrap sample has q marginal features and $\sum_{k=2} \omega \binom{q}{k}$ interaction term.

(2) **Modeling**: fit a model for bootstrap sample i,$i = 1, \ldots, B$ to estimate coefficients β_{ij},$j = 1, \ldots, p, p+1, \ldots, \sum_{k=1}^{\omega} \binom{p}{k}$. The coefficients of unselected features for each bootstrap sample are considered missing.

(3) **Performance measuring**:

(a) compute $(\frac{a}{2}, 1 - \frac{a}{2})$ quantile intervals for the coefficient estimates of all marginal and interaction terms, where $a \in (0,1)$ is a hyperparameter.

(b) compute minimum AUC for each marginal and interaction term based on all bootstrap models that include this term. For term j,j =

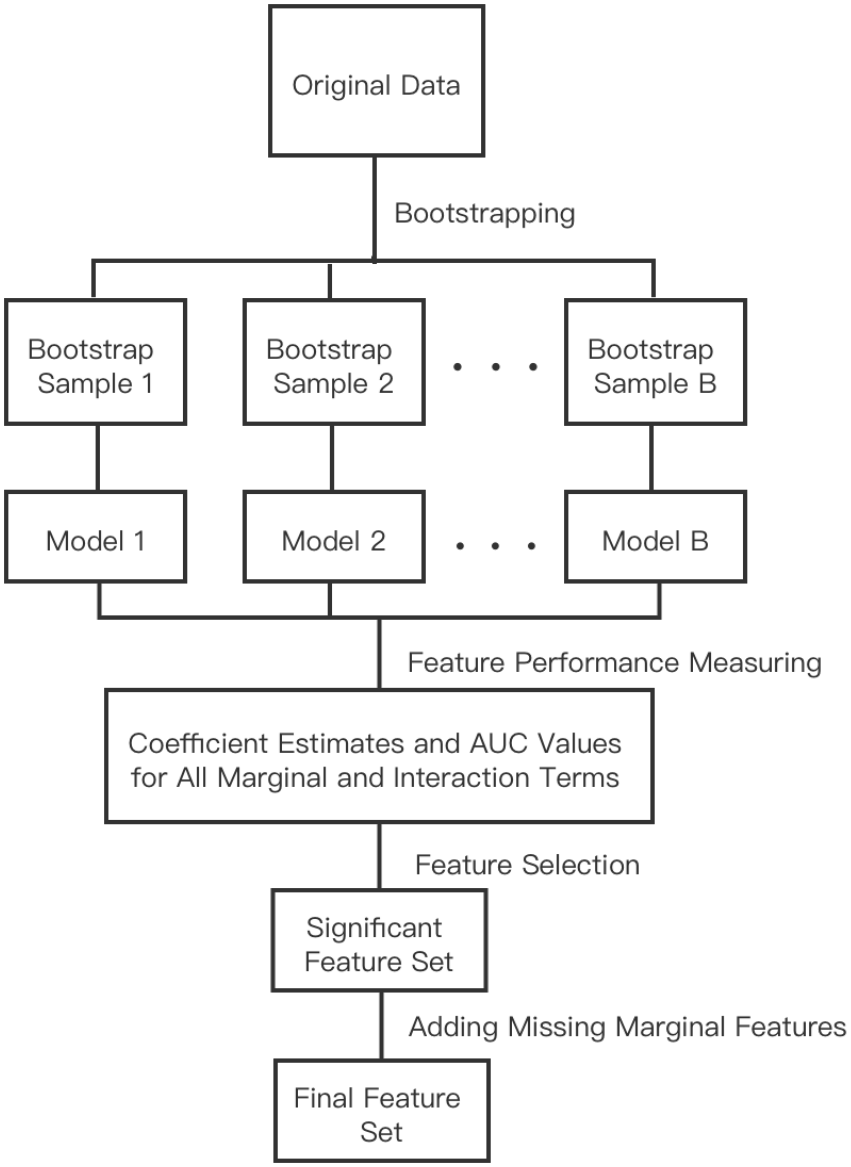

Figure 1: Algorithm of HDSI-BO.

$1, \ldots, p, p+1, \ldots, \sum_{k=1} \omega\binom{p}{k}$,

$$\text{minAUC}_j = \text{minAUC}_{ij} \mid i \in \{1, \ldots, B\}$$

Note that the coefficients and AUC values of unselected features for each bootstrap sample are considered missing. The missing values are dropped during the performance measuring step.

(4) **Feature selection**: select marginal or interaction term j if

(a) the quantile interval does not include 0.

(b) $\text{minAUC}_j > \mu_{\text{minAUC}} + b\sigma_{\text{minAUC}}$,

where

$$\mu_{\text{minAUC}} = \sum_{j=1}^{p} \text{minAUC}_j / p$$

$$\sigma_{\text{minAUC}} = \sqrt{\frac{\sum_{j=1}^{p} \left(\text{minAUC}_j - \mu_{\text{minAUC}}\right)^2}{p-1}}$$

$b \in (-\infty, \infty)$ is a hyperparameter.

(5) **Adding missing marginal features**: Since it's unnatural if a marginal feature is absent in a model while its interaction is present, we add these missing marginal features to obtain a final complete feature set.

Note that any modeling technique, with or without built-in feature selection, can be adopted to model bootstrap samples. This flexibility enables HDSI-BO to combine with multiple techniques such as simple logistic regression, lasso, etc.

2.2 Hyperparameter Optimization

In the HDSI-BO algorithm, there are four hyperparameters whose values need to be predetermined:

(1) number of bootstrap samples B;

(2) number of features for each bootstrap sample q;

(3) hyperparameter a used to determine the bound of quantile intervals;

(4) hyperparameter b used to determine the threshold value for minAUC.

Since the performance of a marginal or interaction term is measured based on its coefficient estimates and values of AUC from different bootstrap models that include this term, it is important to ensure that every term is sampled sufficient times.

HDSI-BO supposes the mean of coefficients are zero and then compares the estimated mean values from bootstrap models with zero. In this case, the minimum number of times a term should be selected L can be determined based on Lehr's equation (Lehr, 2010),

$$L = 8/\Delta^2$$

where Δ is the effect size. According to Cohen's rule of thumb (Cohen, 2013), $\Delta = 0.2$ indicates a small, 0.5 a moderate, and 0.8 a big effect. Smaller effect sizes lead to higher values of L and thus lead to more accurate mean estimates of coefficients.

Suppose ω-level interaction terms are considered, then the probability of

a term to be included in a sample set with q features is

$$\rho = \sum_{k=2}^{\omega} \binom{q}{k} / \sum_{k=2}^{\omega} \binom{p}{k}$$

The B bootstrap samples can be viewed as B trials. Then the times of a term to be selected in B trials $X \sim \text{Binomial}(B,\rho)$ and the probability of one term to be selected at least L times can be calculated as

$$\Pr(X \geq L) = 1 - \sum_{m=0}^{L-1} \Pr(X = m) = f(B,q,L)$$

The probability should be higher enough to ensure almost every term can be selected at least L times. Therefore, the value of B is dependent on q and the effect size if a threshold value of this probability is given.

The optimal values of hyperparameters that can achieve the best accuracy of feature selection and best predictive performance of final models vary with datasets. Therefore, a suitable hyperparameter selection method is needed to determine the optimal values of hyperparameters for given data before the HDSI-BO algorithm can be applied.

Given that there are multiple hyperparameters to be optimized and the distribution of a model's predictive performance based on these hyperparameters is completely unknown, we choose to use the genetic algorithm for the optimization of (q,a,b) in this study (Thede and Scott, 2004). The genetic algorithm mimics the operation of evolution. It generates new combinations of values through mutation and cross-over and searches for the optimal value combinations of hyperparameters that can optimize the performance

of HDSI-BO. To measure the performance of HDSI-BO quantitatively, we fit a logistic regression model based on the selected feature set obtained from HDSI-BO in the training dataset and use the model's AUC in the test dataset as a measure. For the simulation and real data studies below, we use five-fold genetic algorithms with cross-validation. Figure 2 depicts the complete process of applying HDSI-BO in this study.

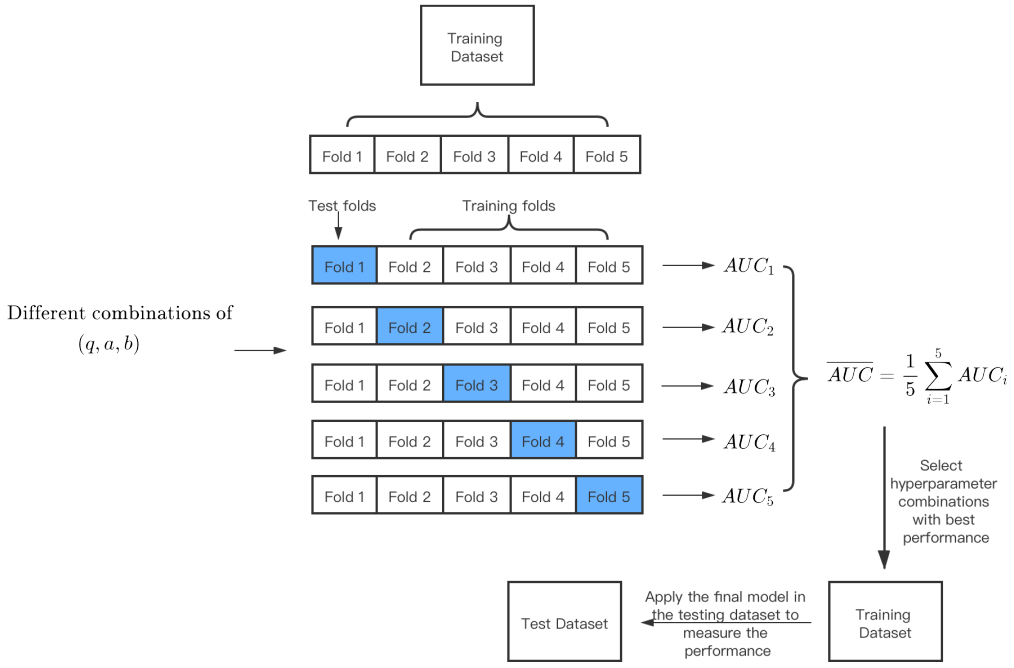

Figure 2: The process of hyperparameter optimization with five-fold genetic algorithm with cross-validation, applying HDSI-BO with selected optimal values of hyperparameters, and finally validating performance.

3 Results

3.1 Simulation Studies

3.1.1 Simulation Settings

To verify the applicability of HDSI-BO in combining with different methods, we combine HDSI-BO with simple logistic regression, lasso, ridge regression, elastic net, adaptive lasso, and carry out simulation studies with two scenarios to compare the predictive performance of HDSI-BO with these methods.

For each scenario, we generate a dataset with a total size of 1000 from a regression model

$$\text{logit}y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \beta_{12} x_1 x_2 + \cdots + \epsilon$$

where $x_1, \cdots, x_p \sim N(0,1)$,$\epsilon \sim N(0,0.01)$.

The sizes of the training set and the testing set are both 300. Stratified sampling is used to ensure that two possible outcomes have equivalent numbers of cases in both sets. For the sake of simplicity, we only take two-way interaction terms into account in the simulation studies although the algorithm itself can incorporate higher-level interactions. The settings of coefficients and feature dimensions are summarized in Table 1.

Table 1: The settings of coefficients and feature dimensions of two simulation scenarios. β_j is the coefficient of X_j and β_{ij} is the two-way interaction term of X_i and X_j. Coefficients not shown in the table are all zero, which indicates the corresponding features are noisy features.

Scenario	Feature Dimension p	Non-zero Coefficients
1	25	$(\beta_0,\beta_1,\beta_2,\beta_3,\beta_4,\beta_{12},\beta_{34}) = (1.8,0.5,0.4,-0.4,0.45,0.6,-0.6)$
2	25	$(\beta_0,\beta_1,\beta_2,\beta_3,\beta_4,\beta_{12},\beta_{34}) = (2.0,0.7,0.6,-0.6,0.65,0.8,-0.9)$

A covariance matrix is designed to add multicollinearity among true and noisy features,

$$\begin{bmatrix} x_1x_1 & x_1x_2 & . & . & x_1x_7 & \cdot & \cdot & \cdot \\ x_2x_1 & x_2x_2 & . & . & x_2x_7 & \cdot & \cdot & \cdot \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & . & . \\ x_6x_1 & x_6x_2 & \cdot & \cdot & x_6x_7 & \cdot & \cdot & \cdot \\ x_7x_1 & x_7x_2 & \cdot & \cdot & x_7x_7 & \cdot & \cdot & \cdot \\ \cdot & \cdot & . & . & \cdot & \cdot & . & . \\ x_px_1 & \cdot & . & . & \cdot & \cdot & . & . \end{bmatrix} = \begin{bmatrix} 1 & 0.3 & 0.3 & 0.6 & 0.6 & 0 & 0 & . & 0 \\ 0.3 & 1 & 0.3 & 0.2 & 0.1 & 0 & 0 & . & 0 \\ 0.3 & 0.3 & 1 & 0.2 & 0.1 & 0 & 0 & . & 0 \\ 0.6 & 0.2 & 0.2 & 1 & 0.1 & 0 & 0 & . & 0 \\ 0.6 & 0.1 & 0.1 & 0.1 & 1 & 0 & 0 & . & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0.1 & . & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 1 & . & 0 \\ . & . & . & . & . & . & . & . & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & . & 1 \end{bmatrix}$$

For both sceanrios, the effect size used for HDSI-BO is large. We adopt genetic algorithm with five-fold cross-validation to optimize hyperparameters (q,a,b) in the training sets and the hyperparameter combination with best mean predictive performance is selected. To measure the predictive performance of different techniques, we use simple logistic regression to fit final models after obtaining sets of selected features and then use AUC of the final models in the testing sets as a measure.

All statistical analysis was performed using software R 4.0.3 in Linux. Sample R codes are provided in the Appendix A

3.1.2 Simulation Results

The feature selection performance of different methods is presented in Table 2. In both scenarios, the five statistical techniques can select all true marginal and interaction terms after being combined with HDSI-BO, which outperforms these standard techniques alone. Note that the standard logistic

regression model cannot include any interaction terms since it cannot work when $p > n$ while HDSI-BO enables it to work under the same condition. Besides, although some methods combined with HDSI-BO tend to select more noisy terms, but it does not hurt the predictive performance of final models. Table 3 shows that methods combined with HDSI-BO all have significantly higher AUC compared with standard methods.

Table 2: Feature selection performance of logistic regression (Reg), lasso, ridge regression (Ridge), elastic net, adaptive lasso alone and after being combined with HDSI-BO.

Scenario		1				2			
Number of Selected Features		Marginal		Interaction		Marginal		Interaction	
		True(4)	Noisy(21)	True(4)	Noisy(298)	True(2)	Noisy(21)	True(2)	Noisy(298)
Standard	Reg	4	21	0	0	4	21	0	0
	Lasso	2	0	0	1	1	0	1	0
	Ridge	4	21	2	296	4	21	2	298
	Elastic Net	2	0	1	12	3	0	1	31
	Adaptive Lasso	3	3	1	72	3	0	1	16
HDSI-BO	Reg	**4**	2	**2**	2	**4**	0	**2**	0
	Lasso	**4**	7	**2**	8	**4**	6	**2**	7
	Ridge	**4**	4	**2**	5	**4**	2	**2**	4
	Elastic Net	**4**	8	**2**	8	**4**	4	**2**	5
	Adaptive Lasso	**4**	14	**2**	18	**4**	13	**2**	21

Table 3: AUC performance of different methods in simulation studies

Scenario	Standard					HDSI-BO				
	Reg	Lasso	Ridge	Elastic Net	Adaptive Lasso	Reg	Lasso	Ridge	Elastic Net	Adaptive Lasso
1	0.686	0.683	0.669	0.688	0.657	0.758	0.755	0.775	0.765	0.756
2	0.732	0.724	0.677	0.705	0.696	0.834	0.816	0.825	0.834	0.834

3.2 Real Data Studies

3.2.1 Real Dataset Summary

We combine the HDSI-BO methods with multiple model fitting techniques and compare their performance with these standard techniques on three real-world datasets. For simplicity, we only take continuous features that do not

have many missing values into account.

For every dataset, we incorporate all two-way interaction terms. A large effect size is used for estimating B. We adopt the same performance measurement as the simulation studies. In addition, 30 trials are repeated for each method in one dataset to generate a range of the number of selected features and 95% confidence interval for AUC.

Dataset I is from National Social Life, health and Aging Project (NSHAP) dataset for Wave 3 (2015-2016) (available at `https://www.icpsr.umich.edu/icpsrweb/NACDA/studies/36873`). The original dataset contains data on 1470 features related to health, social life, and well-being for 4377 older American residents. The original outcome feature is continuous, we convert it into binary by categorizing values above-median as 1 and below-median as 0. Dataset I used for the study has sample size $n = 1633$ and marginal feature dimension $p = 20$.

Dataset II is from Community Health Status Indicators (CHSI) dataset (available at `https://healthdata.gov/dataset/community-health-status-indicators-chsi-combat-obesity-heart-disease-and-cancer`) which contains USA county-level data on various demographics and health parameters. The original dataset contains data on 578 features for 3141 US counties. We convert the self-rated health status score variable into binary by categorizing values above-median as 1 and below-median as 0 for the study. The final dataset we use has a sample size and feature size of 1470 and 29, respectively.

Dataset III is the Breast Cancer Coimbra dataset (available at `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Coimbra`). It contains 9 quantitative predictors, which are anthropometric data and parameters gathered in routine blood analysis and a binary dependent variable, indicating the presence or absence of breast cancer. The sample size is 116.

Table 4 presents the summary of these real datasets. Each dataset is divided into training and test sets.

Table 4: Summary of real datasets

Dataset	Outcome feature	Sample size(n)			Marginal features(p)
		Total	Train	Test	
I	Height	1633	1306	327	20
II	Health status	1470	1176	294	29
III	Breast cancer	116	93	23	9

3.2.2 Results

Table 5 and Table 6 summarize the results of real data studies. In most cases, the HDSI-BO methods select less marginal and interaction terms while the mean values of AUC of the HDSI-BO methods are close or higher than those of the standard methods, which indicates that the HDSI-BO methods have comparable or better predictive ability compared with existing standard methods.

Table 5: Feature selection performance of different methods. Ranges are generated based on 30 trials.

Method		Dataset					
		I		II		III	
		Marginal(20)	Interaction(190)	Marginal(29)	Interaction(406)	Marginal(9)	Interaction(36)
Standard	Reg	20(20-20)	19(190-190)	29(29-29)	406(406-406)	9(9-9)	36(36-36)
	Lasso	19(16-20)	28(11-64)	11(4-16)	9(1-30)	4(1-6)	6(1-13)
	Ridge	20(20-20)	190(190-190)	29(29-29)	406(406-406)	9(9-9)	36(36-36)
	Elastic Net	20(16-20)	65(8-152)	18(13-20)	42(0-106)	6(3-9)	12(3-32)
	Adaptive Lasso	18(15-20)	24(9-83)	14(5-20)	11(0-43)	4(2-6)	4(2-8)
HDSI-BO	Reg	14(8-16)	12(7-16)	15(10-21)	15(8-26)	5(2-7)	3(1-5)
	Lasso	19(15-20)	24(16-30)	16(8-23)	17(8-26)	7(3-9)	7(2-9)
	Ridge	14(10-16)	18(14-23)	22(18-26)	28(17-36)	4(2-6)	3(1-4)
	Elastic Net	16(12-19)	21(13-28)	17(14-24)	16(10-21)	5(2-9)	4(1-9)
	Adaptive Lasso	18(13-20)	28(19-39)	8(3-14)	6(2-13)	6(4-9)	4(2-8)

Table 6: AUC performance of different methods. The values presented in the table are the average and 95% confidence intervals of AUC over 30 trials.

Method		AUC (95% CI)		
		I	II	III
Standard	Reg	0.748(0.680-0.817)	0.940(0.919-0.961)	0.648(0.481-0.814)
	Lasso	0.785(0.746-0.824)	0.925(0.903-0.946)	0.784(0.537-1)
	Ridge	0.747(0.706-0.788)	0.911(0.891-0.945)	0.777(0.585-0.969)
	Elastic Net	0.783(0.739-0.828)	0.930(0.907-0.953)	0.761(0.531-0.990)
	Adaptive Lasso	0.786(0.740-0.833)	0.927(0.905-0.949)	0.793(0.588-0.998)
HDSI-BO	Reg	0.799(0.749-0.834)	0.925(0.893-0.956)	0.810(0.625(0.995)
	Lasso	0.802(0.756-0.847)	0.930(0.905-0.955)	0.792(0.609-0.974)
	Ridge	0.793(0.745-0.841)	0.930(0.904-0.956)	0.793(0.610-0.976)
	Elastic Net	0.797(0.750-0.844)	0.931(0.909-0.953)	0.782(0.591-0.972)
	Adaptive Lasso	0.800(0.758-0.842)	0.919(0.887-0.951)	0.787(0.576-0.997)

4 Conclusion and Discussion

In this paper, we introduced a new high-dimensional data feature selection algorithm for binary outcome data (HDSI-BO). It takes interaction terms into account and can incorporate with existing modeling techniques.

Based on the results of the simulation studies, the HDSI-BO methods can select more true features than the standard methods. Although the HDSI-BO methods selected more noisy features than standard methods in some cases, the values of AUC for the HDSI-BO methods are still higher than those of the standard methods in most cases. It indicates that these noisy features do not significantly damage the predictive ability of the models. Therefore, it is acceptable that HDSI-BO methods select some noisy features. The real data studies showed similar results in terms of the predictive ability of the HDSI-BO methods. In summary, the HDSI-BO algorithm, combined with the statistical techniques used in the simulation and real data studies, performs better in selecting true features and consequently, in prediction accuracy compared with the standard techniques.

In addition to the potential of HDSI-BO in doing feature selection and prediction, the current studies also show limitations. First, the existing feature selection criteria can be modified and improved to reduce the selection of noisy features. Second, AUC, as the single metric of performance, cannot comprehensively measure the feature selection performance of the HDSI-BO methods. Finally, the HDSI-BO algorithm has not been combined with non-

linear methods such as random forest, support vector machine. Such extension is important to further verify the applicability. Future work can focus on overcoming the limitations mentioned above and extending the applications of HDSI-BO.

Acknowledgments

I would like to express my sincere gratitude to my practicum supervisor Dr. Wei Xu and co-supervisor Dr. Rahi Jain for the continuous support during my practicum. Their guidance helped me in all the time of research and writing of this report.

I am also grateful to Professor Tony Panzarella and all professors who contributed towards the practicum course. Because of them, I was able to work on this project and make it good and enjoyable experience.

References

[1] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Academic press, 2013.

[2] Frank Emmert-Streib and Matthias Dehmer. High-dimensional lasso-based computational regression models: regularization, shrinkage, and selection. *Machine Learning and Knowledge Extraction*, 1(1):359–383, 2019.

[3] Lance W Hahn, Marylyn D Ritchie, and Jason H Moore. Multifactor dimensionality reduction software for detecting gene–gene and gene–environment interactions. *Bioinformatics*, 19(3):376–382, 2003.

[4] Georg Heinze, Christine Wallisch, and Daniela Dunkler. Variable selection–a review and recommendations for the practicing statistician. *Biometrical journal*, 60(3):431–449, 2018.

[5] Rahi Jain and Wei Xu. HDSI: High dimensional selection with interactions algorithm on feature selection and testing. *PloS one*, 16(2):e0246159, 2021.

[6] Jian Kang and Jian Guo. Self-adaptive lasso and its bayesian estimation. Technical report, Working Paper, 2009.

[7] R. Lehr. Sixteen s-squared over d-squared: a relation for crude sample size estimates. *Statistics in Medicine*, 11(8):1099–1102, 2010.

[8] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas. *arXiv preprint arXiv:1912.06059*, 2019.

[9] Scott M Thede. An introduction to genetic algorithms. *Journal of Computing Sciences in Colleges*, 20(1):115–123, 2004.

[10] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.

Appendix A

Below is the code for running genetic algorithm to optimize hyperparameters.

```
#Set working directory

#Load packages and functions
library(memoise)
library(glmnet)
library(ROCR)
library(Rmisc)
library(GA)
library(mosaic)
library(parallel)
library(doParallel)
library(caret)
library(msgps)
library(pROC)
library(plyr)
#Read parameter values from the bash script
args=commandArgs(T)
scenario=args[1]
folder=args[2]
method=args[3]
k=as.numeric(args[4])
sapply(list.files("HDSI-code",pattern="*.R$"), function(x)
    source(paste0("HDSI-code/",x)))
#read training dataset and test dataset
if (folder=="realdata") {
sample<-readRDS(file= paste0("output/realdata",scenario,"/
    sample1.RData"))
traindf<-sample$train
} else {
traindf<-readRDS(file= paste0("output/scenario",scenario,"/
    traindf1.RData"))
}
#Generate formula & design matrix
int_term=2
x<-c(rep(".*",int_term-1),".")
x<-Reduce('paste0',x)
f=as.formula(paste("y ~",x))
Matrix=model.matrix(f,traindf)[,-1] #model.matrix: create a
    design matrix
outvar="y"

```

```
#####################################
##        Genetic Algorithm        ##
#####################################
GA_auc<-ga(type="real-valued",fitness = fitness,train=traindf,
    effectsize="large",rule="auc",method=method,lower=c
    (2,0.8,0.5),upper=c(k,1,2),popSize=20,run=10,maxiter=200,
    parallel=T,pcrossover=0.8,pmutation=0.4,nfold=5)
summary(GA_auc)
capture.output(summary=summary(GA_auc),file=paste0("output/",
    folder,scenario,"/results/GA_",method,"_auc.txt"),append=T)
```

Below is the code for running the HDSI-BO algorithm

```
HDSI_binary<-function(df,int=interaction_numb,outvar="y",method=
    c("glm","forward","lasso","ridge","elastic","Alasso"),
    effectsize="large",seed=1,q=NA,a,b,selection=c("auc","bic")){
  #read training dataset and testdataset
  traindf<-df$train
  testdf<-df$test
  #Generate formula & design matrix
  f<-gen_formula(int=int,outvar=outvar)
  Matrix=model.matrix(f,traindf)[,-1] #model.matrix: create a
      design matrix
  #Bootstraps
  boots<-mbootsample(k=q,interaction_numb=int, effectsize=
      effectsize, inputdf=traindf, outvar=outvar,seed_multiplier=
      seed)

  #Fit models with HDSI methods
  methodlist=list(glm = HDSI_regression, lasso = HDSI_lasso,
      forward = HDSI_forward,ridge=HDSI_ridge,elastic=HDSI_
      elastic,Alasso=HDSI_Alasso)
  #op <-pbapply::pboptions(nout=9000) #the maximum number of
      times the progress bar is updated
  result=lapply(1:length(boots), function(x){
    rows=boots[[x]][[2]]      #samples
    columns=boots[[x]][[1]]   #features
    df=traindf[rows, columns]
    y<-traindf[rows,outvar]
    df<-cbind(df,y)
    # Run the model
    return(methodlist[[method]](traindf = df, outvar = outvar, f
        = f,boot=x))
  })
```

```
  #pbapply::pboptions(op)

  #Summarize the final results
  res<-rbind.fill(fun1(result,1),all=T)
  auc<-rbind.fill(fun1(result,2),all=T)
  bic<-rbind.fill(fun1(result,3),all=T)
  #Feature selection
  feature<-feature_selection(res,auc,bic,a=a,b=b,selection=
      selection)
  split_var = unlist(strsplit(feature, ":"))
  feature.new<-unique(c(split_var,feature))
  interaction<-sum(grepl(":",feature.new))
  marginal<-length(feature.new)-interaction
  if(length(feature.new)==0) return(list(summary=data.frame(
      marginal=0,interaction=0,auc=0)))

  #Fit new models
  f.new=as.formula(paste(outvar,"~", paste(feature.new, collapse
       = "+")))
  glm.new <- glm(f.new,family = binomial(link = logit), data =
      traindf,maxit = 200)
  auc.fin<-auc(testdf[,outvar],as.vector(predict(glm.new,testdf,
      type="response")))
  fin<-list(summary=data.frame(marginal=marginal,interaction=
      interaction,auc=auc.fin),features=feature.new)
  return(fin)
}
```

Below are functions used for the genetic algorithm and the HDSI-BO algorithm.

```
#fitness function
fitness = function(x,nfold=3,train=traindf,interaction_numb=2,
    outvar="y",method="glm",effectsize="large",seed=1,rule=c("auc
    ","bic")){
  q<-ceiling(x[1])
  datasize<-nrow(train)
  index<-CVgroup(K=nfold,datasize=datasize,seed=seed)
  #cl<-makeCluster(39,type="FORK")
  res<-sapply(1:nfold,K_HDSI,data=train,index=index,interaction_
      numb=interaction_numb,method=method,effectsize=effectsize,
      seed=seed,q=q,a=x[2],b=x[3],selection=rule,outvar=outvar)
  #stopCluster(cl)
```

```
  mean_auc<-mean(res)
  return(mean_auc)
}

#Split K groups
CVgroup <- function(K,datasize,seed){
  cvlist <- list()
  set.seed(seed)
  n <- rep(1:K,ceiling(datasize/K))[1:datasize]
  temp <- sample(n,datasize)
  x <- 1:K
  dataseq <- 1:datasize
  cvlist <- lapply(x,function(x) dataseq[temp==x])
  return(cvlist)
}

#K-fold Cross-validation
K_HDSI = function(x,data=train,index=index,interaction_numb=
    interaction_numb,method=method,effectsize=effectsize,seed=
    seed,q=q,a=a,b=b,selection,outvar=outvar){
  traindf<-data[-index[[x]],]
  testdf<-data[index[[x]],]
  df<-list(train=traindf,test=testdf)
  return(HDSI_binary(df=df,int=interaction_numb,outvar=outvar,
      method=method,effectsize=effectsize,seed=seed,q=q,a=a,b=b,
      selection=selection)$summary$auc)
}
#Calculate AUC
CalAUC =function(model,testdf,real){
  pred=predict(model,newdata = testdf,type='response')
  rocr.pred=prediction(pred,real)
  rocr.perf=performance(rocr.pred,'auc')
  as.numeric(rocr.perf@y.values)
}

#Feature selection
feature_selection=function(coef,auc,bic,a,b,selection=c("auc","
    bic")){
  ind<-apply(coef,2,function(x){
    x<-x[!is.na(x)]
    mean_x<-mean(x)
    sd_x<-sd(x)
    #x_lower<-mean_x-1.96*sd_x
    #x_upper<-mean_x+1.96*sd_x
    q_low<-quantile(x,(1-a)/2)
```

```
    q_upper<-quantile(x,1-(1-a)/2)
    a<-ifelse((q_low<0 & q_upper>0)| is.na(sd_x),0,1)})
  coef_name<-names(ind)[ind!=0]
  #auc threshold,b=
  #max.auc<-apply(auc,2,function(x){
  #  x<-x[!is.na(x)]
  #  if(length(x)==0) return(NA)
  #  else
  #  return(max(x,na.rm=T))})
  #max.auc<-max.auc[-1]
  #max.auc<-max.auc[!is.na(max.auc)]
  #mean_auc<-mean(max.auc)
  #sd_auc<-sd(max.auc)
  #auc_upper<-mean_auc+1.64*sd_auc
  #y<-names(max.auc)[max.auc>=quantile(max.auc,0.9)]
  if (selection=="auc") {
  min.auc<-apply(auc,2,function(x){
    x<-x[!is.na(x)]
    if(length(x)==0) return(NA)
    else
      return(min(x,na.rm=T))})
  min.auc<-min.auc[-1]
  min.auc<-min.auc[!is.na(min.auc)]
  mean.min_auc<-mean(min.auc)
  sd.min_auc<-sd(min.auc)
  auc_lower<-mean.min_auc+b*sd.min_auc
  auc_name<-names(min.auc)[min.auc>auc_lower]
  comb<-intersect(coef_name,auc_name)
  return(comb)
  }else{
    max.bic<-apply(bic,2,function(x){
      x<-x[!is.na(x)]
      if(length(x)==0) return(NA)
      else
        return(max(x,na.rm=T))})
    max.bic<-max.bic[-1]
    max.bic<-max.bic[!is.na(max.bic)]
    mean.max.bic<-mean(max.bic)
    sd.max.bic<-sd(max.bic)
    bic_upper<-mean.max.bic+b*sd.max.bic
    bic_name<-names(max.bic)[max.bic<bic_upper]
    comb<-intersect(coef_name,bic_name)
    return(comb)
  }
}
```

```
#Generate formula
gen_formula=function(int,outvar){
  x<-c(rep(".*",int-1),".")
  x<-Reduce('paste0',x)
  f<-as.formula(paste(outvar," ~ ",x))
  return(f)
}

#Create result list
make_result = function(type=c("coef","auc"),method=method,n=
    length(boots),s=scenario){
  if(!require(plyr)) install.packages("plyr")
  library(plyr)
  res<-lapply(1:n,function(x) readRDS(file= paste0("output/
      scenario",s,"/",method,x,".RData"))[[type]])
  res<-rbind.fill(res,all=T)
  return(res)
}

#Make result list - when no RData is saved
fun1 <- function(lst, n){
  sapply(lst, `[`, n)
}

#BIC
bic_cal<-function(model){
tLL <-  - deviance(model)
k <- model$df
n <- model$nobs
#AICc <- -tLL+2*k+2*k*(k+1)/(n-k-1)
#AICc

BIC<-log(n)*k - tLL
return(BIC)
}

#sampling data for simulation scenarios
gendata<-function(ta){
  index_0<-which(ta$y==0)
  index_1<-which(ta$y==1)
  index_0<-sample(index_0,150,replace=F)
  index_1<-sample(index_1,150,replace=F)
  index<-c(index_0,index_1)
  return(ta[index,])
```

```
}
#Regression for HDSI-BO
HDSI_regression=function(traindf = df, outvar = outvar, f = f,
    boot){
  x=traindf[,names(traindf)!=outvar]
  y=traindf[,outvar]

  glm <- glm(f,family = binomial(link = logit), data = traindf,
      maxit = 200)
  coef.glm<-as.data.frame(t(glm$coefficients))
  auc.ord<-auc(y,predict(glm,traindf,type="response"))
  aucdata.glm<-as.data.frame(t(glm$coefficients))
  aucdata.glm[]<-auc.ord
  bic<-BIC(glm)
  bic.glm<-as.data.frame(t(glm$coefficients))
  bic.glm[]<-bic
  res<-list(coef=coef.glm,auc=aucdata.glm,bic=bic.glm)
}
#LASSO for HDSI-BO
HDSI_lasso=function(traindf=df,outvar=outvar,f=f,boot){
  Matrix=stats::model.matrix(f,traindf)[,-1]
  cvfit=cv.glmnet(Matrix,traindf[,outvar],family="binomial",
      alpha=1,type.measure="class")
  lambda.1se=cvfit$lambda.1se
  lambda.min=cvfit$lambda.min
  model=glmnet(Matrix, traindf[,outvar], lambda = lambda.1se,
      alpha=1, standardize=F, family="binomial")
  coef=as.data.frame(t(as.matrix(coef(model,s=lambda.1se))))
  if(sum(coef!=0)<=1){ #test if the model with lambda.1se is too
      simple (only no parameters)
    model = glmnet::glmnet(Matrix, traindf[,outvar], lambda =
        lambda.min, alpha=1, standardize=F, family="binomial")
    coef=as.data.frame(t(as.matrix(coef(model,s=lambda.min))))
  }
  lasso.pred=predict(model,newx=Matrix,type="response")
  pred<- prediction(lasso.pred,traindf[,outvar])
  auc.perf<-performance(pred,measure="auc")
  auc.ord<-unlist(auc.perf@y.values)
  aucdata.lasso<-coef
  aucdata.lasso[aucdata.lasso!=0]<-unlist(auc.ord)
  aucdata.lasso[aucdata.lasso==0]<-NA
  coef[coef==0]<-NA
  bic<-bic_cal(model)
  bic.lasso<-coef
  bic.lasso[bic.lasso!=0]<-bic
```

```
  bic.lasso[bic.lasso==0]<-NA
  res<-list(coef=coef,auc=aucdata.lasso,bic=bic.lasso)
  #res<-list(coef=coef,auc=aucdata.lasso)
  #saveRDS(res, file= paste0("output/scenario",scenario,"/lasso",boot,".RData"))
}
#Elastic net for HDSI-BO
HDSI_elastic=function(traindf=df,outvar=outvar,f=f,boot){
  Matrix=stats::model.matrix(f,traindf)[,-1]
  elastic<-train(f,data=traindf,method="glmnet",trControl=trainControl("cv",number=5),tuneLength=10)
  alpha<-elastic$bestTune[1]
  lambda<-elastic$bestTune[2]
  model=glmnet(Matrix, traindf[,outvar], lambda = lambda, alpha=alpha, standardize=F, family="binomial")
  coef=as.data.frame(t(as.matrix(coef(model,s=lambda))))
  coef[coef==0]<-NA
  elastic.pred=predict(model,newx=Matrix,type="response")
  pred<- prediction(elastic.pred,traindf[,outvar])
  auc.perf<-performance(pred,measure="auc")
  auc.ord<-unlist(auc.perf@y.values)
  aucdata.elastic<-coef
  aucdata.elastic[!is.na(aucdata.elastic)]<-unlist(auc.ord)
  bic<-bic_cal(model)
  bic.elastic<-coef
  bic.elastic[!is.na(coef)]<-bic
  res<-list(coef=coef,auc=aucdata.elastic,bic=bic.elastic)
}
#Ridge regression for HDSI-BO
HDSI_ridge=function(traindf=df,outvar=outvar,f=f,boot){
  Matrix=stats::model.matrix(f,traindf)[,-1]
  cvfit=cv.glmnet(Matrix,traindf[,outvar],family="binomial",alpha=0,type.measure="class")
  lambda.1se=cvfit$lambda.1se
  lambda.min=cvfit$lambda.min
  model=glmnet(Matrix, traindf[,outvar], lambda = lambda.1se, alpha=0, standardize=F, family="binomial")
  coef=as.data.frame(t(as.matrix(coef(model,s=lambda.1se))))
  if(sum(coef!=0)<=1){ #test if the model with lambda.1se is too simple (only no parameters)
    model = glmnet::glmnet(Matrix, traindf[,outvar], lambda = lambda.min, alpha=0, standardize=F, family="binomial")
    coef=as.data.frame(t(as.matrix(coef(model,s=lambda.min))))
  }
  lasso.pred=predict(model,newx=Matrix,type="response")
```

```
  pred<- prediction(lasso.pred, traindf[,outvar])
  auc.perf<-performance(pred, measure="auc")
  auc.ord<-unlist(auc.perf@y.values)
  aucdata.lasso<-coef
  aucdata.lasso[aucdata.lasso!=0]<-unlist(auc.ord)
  aucdata.lasso[aucdata.lasso==0]<-NA
  coef[coef==0]<-NA
  bic<-bic_cal(model)
  bic.lasso<-coef
  bic.lasso[!is.na(bic.lasso)]<-bic
  res<-list(coef=coef,auc=aucdata.lasso,bic=bic.lasso)
  #res<-list(coef=coef,auc=aucdata.lasso)
  #saveRDS(res, file= paste0("output/scenario",scenario,"/lasso
      ",boot,".RData"))
}
#Adaptive lasso for HDSI-BO
HDSI_Alasso=function(traindf=df,outvar=outvar,f=f,boot){
  Matrix=stats::model.matrix(f,traindf)[,-1]
  Alasso<-cv.glmnet(Matrix,traindf[,outvar],family="binomial",
      alpha=0,type.measure="class")
  lambda.min<-Alasso$lambda.min
  best_ridge_coef<-as.numeric(coef(Alasso,s=lambda.min))[-1]
  alasso_cv<-cv.glmnet(Matrix,traindf[,outvar],family="binomial"
      ,alpha=1,penalty.factor=1/abs(best_ridge_coef),type.measure
      ="class")
  alasso<-glmnet(Matrix,traindf[,outvar],family="binomial",alpha
      =1,penalty.factor=1/abs(best_ridge_coef),lambda=alasso_cv$
      lambda.1se)
  Coef.alasso=as.data.frame(t(as.matrix(coef(alasso_cv,s=alasso_
      cv$lambda.1se))))
  if(sum(Coef.alasso!=0)<=1){ #test if the model with lambda.1se
       is too simple (only no parameters)
    alasso<-glmnet(Matrix,traindf[,outvar],family="binomial",
        alpha=1,penalty.factor=1/abs(best_ridge_coef),lambda=
        alasso_cv$lambda.min)
    Coef.alasso=as.data.frame(t(as.matrix(coef(alasso_cv,s=
        alasso_cv$lambda.min))))
  }
  Coef.alasso[Coef.alasso==0]<-NA
  alasso.pred=predict(alasso,newx=Matrix,type="response")
  auc<-auc(traindf[,outvar],as.vector(alasso.pred))
  #rownames(Coef.alasso)[Coef.alasso!=0]
  aucdata.alasso<-Coef.alasso
  aucdata.alasso[!is.na(aucdata.alasso)]<-auc
  bic<-bic_cal(alasso)
```

```
  bic.alasso<-Coef.alasso
  bic.alasso[!is.na(bic.alasso)]<-bic
  res<-list(coef=Coef.alasso,auc=aucdata.alasso,bic=bic.alasso)
}
#Estimate the number of boostrap samples B
B_est=function(p,k=NA,rows=NA, interaction_numb=2, effectsize= c
    ("large", "medium", "small")){
  #print(c(p,k))
  denominator=choose(p,interaction_numb)
  numerator=choose(k,interaction_numb) # where, 2 is the order
      of interaction eg of 2 order interaction is X1_X2
  #print(c(numerator,denominator))
  init_prob=numerator/denominator

  # Minimum bootstraps which will give minimum 13 occurrences (
      consider large effect) of an interaction variable with 99%
      confidence
  if(effectsize=="large"){prefer_occurence = 13}
  else if (effectsize == "medium"){prefer_occurence = 32}
  else if(effectsize == "small"){prefer_occurence = 200}
  else {prefer_occurence = effectsize}
  # print(c(prefer_occurence,init_prob))
  min_boot= ceiling(prefer_occurence/init_prob) # where 32 is
      minimum number of occurence desired for an interaction
      variable during bootstrapping.
  #print(min_boot)
  max_boot= 8*min_boot
  #print(max_boot)
  f_opt=function(x,y=init_prob,max_x=max_boot,prefer=prefer_
      occurence){
    Val=qbinom(0.05, floor(x), y) #P(X>Val)>=95%
    value=(prefer-Val)^2+(x/max_x)
    return(value)
  }
  bootvalue = optimize(f_opt,interval = c(min_boot,max_boot))
  # print(ceiling(bootvalue$minimum))
  return(ceiling(bootvalue$minimum))
}

#Bootstrapping
bootsample=function(k=NA,interaction_numb=2, effectsize="large",
    inputdf, outvar,seed_multiplier=1){
  p<-dim(inputdf)[2]-length(outvar)
  rows<-dim(inputdf)[1]
  # Find k if not provided by user
```

```
285   if(is.na(k)==T){
        optimal_k=function(x, r=rows){
287       int=choose(x,interaction_numb)
          value=abs((int+x)-r)/r
289       return(value)
        }
291     k=stats::optimize(optimal_k,interval = c(interaction_numb,p)
            )$minimum
        k=floor(k)
293   }

295   #print(p)
      # Find the number of sample that need to be created
297   boots=B_est(p=p,k=k, rows=rows, interaction_numb=interaction_
          numb, effectsize=effectsize)
      #cat(c("Bootstraps:",boots))
299
      ## Create Variable list for each bootstrap
301   res=bootsample_binary(boots=boots, p=p,rows=rows, k=k, seed_
          multiplier = seed_multiplier,inputdf=inputdf,outvar=outvar)

303   return(res)
}
305
bootsample_binary=function(boots=boots, p, rows, k, seed_
    multiplier=1, inputdf,outvar){
307   if(!require(pbapply)) install.packages("pbapply")
      library(pbapply)
309   ## Create Variable list for each bootstrap
      samples=1:rows
311   op <- pbapply::pboptions(type = "timer") #adds progress bar to
          vectorized R functions
      res=pblapply(1:boots, function(x) {
313     set.seed(x*seed_multiplier)
        ### Create variable combinations
315     features <- sample(p, k, replace = FALSE) # replaced samples
            with features/3
        features<- sort(features)
317     ## Create Sample list for each bootstrap
        CAT_1 <- which(inputdf[,outvar]==1)
319     CAT_0 <- which(inputdf[,outvar]==0)
        S_1<-sample(CAT_1,length(CAT_1),replace=T)
321     S_0<-sample(CAT_0,length(CAT_0),replace=T)
        samples <- c(S_1,S_0)
323     list(features, samples)
```

```
  })
  pbapply::pboptions(op)
  return(res)
}
mbootsample = memoise::memoise(bootsample)
```