Journal of Statistical Software

September 2017, Volume 80, Issue 7. Reddoi: 10.18637/jss.v000.i00

Feature Selection with the R Package MXM: Discovering Statistically-Equivalent Feature Subsets

Vincenzo Lagani
University of Crete

Giorgos Athineou
University of Crete

Alessio Farcomeni
Sapienza - University of Rome

Michail Tsagris
University of Crete

Ioannis Tsamardinos
University of Crete

Abstract

The statistically equivalent signature (SES) algorithm is a method for feature selection inspired by the principles of constrained-based learning of Bayesian Networks. Most of the currently available feature-selection methods return only a single subset of features, supposedly the one with the highest predictive power. We argue that in several domains multiple subsets can achieve close to maximal predictive accuracy, and that arbitrarily providing only one has several drawbacks. The SES method attempts to identify multiple, predictive feature subsets whose performances are statistically equivalent. Under that respect SES subsumes and extends previous feature selection algorithms, like the max-min parent children algorithm.

SES is implemented in an homonym function included in the R package **MXM**, standing for mens ex machina, meaning 'mind from the machine' in Latin. The **MXM** implementation of SES handles several data-analysis tasks, namely classification, regression and survival analysis. In this paper we present the SES algorithm, its implementation, and provide examples of use of the `SES` function in R. Furthermore, we analyze three publicly available data sets to illustrate the equivalence of the signatures retrieved by SES and to contrast SES against the state-of-the-art feature selection method LASSO. Our results provide initial evidence that the two methods perform comparably well in terms of predictive accuracy and that multiple, equally predictive signatures are actually present in real world data.

Keywords: feature selection, constraint-based algorithms, multiple predictive signatures.

1. Introduction

Feature selection is one of the fundamental tasks in the area of machine learning. Generally speaking, the process of feature or variable selection aims to identify a subset of features that

are relevant with respect to a given task; for example, in regression and classification it is often desirable to select and retain only the subset of variables with the highest predictive power. The main goals of feature selection usually are (a) to improve the performance of a predictive model, (b) to avoid the cost associated with measuring all the features and (c) to provide a better understanding of the predictive model, and by extension of the data, by eliminating useless or redundant features (**?**). To date, almost all feature selection algorithms return a single feature subset.

In our experience, it is often the case that multiple feature subsets are approximately equally predictive for a given task. Low statistical power due to an insufficient sample size can simply make it impossible to distinguish the predictive performance of two or more signatures in a statistically meaningful way. More intriguingly, the physical process that generates the data could be possibly characterized by a high level of redundancy: several of its components can have similar or identical behavior/scope. Measurements taken over redundant components would be equivalent to each other, and there would be no particular reason for preferring one over the other for inclusion in a predictive subset. This problem is particularly relevant in biology, where nature uses redundancy for ensuring resilience to shocks or adverse events.

Discovering multiple and statistically equivalent feature subsets has several advantages in our opinion. First, knowing that multiple equally-predictive subsets actually exist increases the understanding of the specific problem at hand. In contrast, identifying a single subset of relevant features can lead to ignore factors that may play an important role for understanding the dynamics of the problem under study. On more practical terms, equally-predictive subsets may differ in terms of the cost/effort needed for measuring their respective components. Thus, providing multiple, alternative subsets can have a great impact in contexts where some factors may be technically difficult or excessively expensive to measure.
Recently, algorithms that generate multiple, equivalent feature sets have been developed (**??**), including the Statistically Equivalent Signatures (SES) method (**?**), which is implemented in the R (**?**) **MXM** package. SES is a constraint-based, feature selection algorithm that attempts to identify multiple, equally-predictive *signatures*, where for signatures we indicate minimal-size sets of features with maximal predictive power. SES subsumes and extends previous work on future selection, particularly the max-min parent children (MMPC) algorithm (**?**) and related extensions (**??**), by implementing a heuristic method for identifying equivalences among predictors.
Other statistical approaches producing several models for the same task exist, for example model averaging (**?**). In this approach several competitive models are first generated and then combined together for producing a single, global model. The key difference with SES is that model-averaging models can have different predictive capabilities, while SES-retrieved signatures are assumed to be equally predictive. Model averaging methods are already available in several R packages, like **MuMIn** (**?**), **glmulti** (**?**), **AICcmodavg** (**?**), **BMA** (**?**).
Finally, to the best of our knowledge, the **MXM** package is one of the few open-source code providing implementations of constraint-based feature selection algorithms. The MMPC algorithm has been previously implemented in the **bnlearn** package (**?**) along with several Bayes Network learning methods, and the TETRAD software (**?**) provides implementations of numerous casual-discovery oriented constraint-based methods. The MATLAB library Causal Explorer (**?**) has been the first software offering feature-selection oriented constraint-based methods, but the code is not open-source.
In the rest of the paper we present the SES algorithm and detail its characteristics. Moreover,

we introduce the **MXM** package and provide some practical examples for illustrating its use. Finally, we empirically evaluate the results of the SES algorithm on three different data sets, and we contrast our results against the widely used LASSO selection algorithm (**?**). Our results support claims that SES is able to return signatures that are statistically equivalent, and whose predictive performances are comparable with the ones of a state-of-the-art feature selection method.

2. Multiple signature selection with SES algorithm

The SES algorithm (**?**) belongs to the class of constraint-based, feature selection algorithms (**?**), a class of algorithms that ground their root in the theory of Causal Analysis (**?**). Principles borrowed from this theory allow for an important result: under some broadly-accepted assumptions, the optimal set of predictors for a given target variable consists in the Markov Blanket (MB) of the variable in the Bayesian Network (BN) representing the data distribution at hand (**?**). Bayesian Networks (**?**) are graphical models that allow compact representations of multivariate distributions under the form of a Direct Acyclic Graph (DAG) and an appropriate parameterization. Nodes in the DAG represent random variables, while edges represent conditional associations. When two nodes are directly connected by an edge, then the association between the two corresponding variables holds in the context of all other variables. Node A is a parent for node B (and B is a child of A) if an edge from A is incident to B. The MB of a given target T in composed by the set of Parent and Children (PC) of T plus any additional parent of T children (spouses). MMPC was one of the first feature selection methods specifically designed in order to identify the PC set of a given variable. It is interesting to note that PC and BN predictive capabilities are often equivalent in practical applications, while PC is easier to identify (**?**). Finally, constraint-based algorithms have recently proven to be able to retrieve highly predictive signatures (**?**).

From an algorithmic point of view, given a data set D defined over a set of n variables / predictors V and a target variable T (a.k.a. outcome), constraint-based feature selection methods repetitively apply a statistical test of conditional independence in order to identify the subset of variables that can not be made independent by the outcome given any other subset of variables in V. We denote with $ind(X, T|W)$ any statistical test able to provide a p value $p_{XT.W}$ for assessing the null hypothesis that the variables X and T are conditionally independent given a set of variables W. Depending on the nature of the variables involved in the test (e.g., categorical, continuous, censored) the most appropriate conditional independence test must be chosen (see Section **??** for further discussion). Finally, it is worthwhile to note that under some additional assumptions, constraint-based methods have the interesting property of uncovering (part of) the causal mechanism that produced the data at hand.

The SES algorithm also retrieves the PC set of target variables, and it subsumes the MMPC algorithm by implementing an additional heuristic in order to retrieve multiple subsets that are possible PC candidates. The iTIE* algorithm (**?**) is based on similar principles, but it adopts a different heuristic for identifying equivalence of features. SES is summarized in Algorithm **??** through pseudo code. The proposed method accepts as input a data set D, a target variable T, a two hyper-parameters[1]: The threshold a for assessing conditional independence

[1] we define as "hyper-parameter" any parameter that is not estimated from the data but that must be provided a priori from the user.

and an upper bound k on the number of variables that can be included in any conditional set. This latter parameter limits the complexity and the computational requirements of the algorithm. The output of the method consists in a set E of variables sets (queues) Q_i, $i = 1 \dots n$, such that each queue Q_i contains variables that are 'equivalent' to each other.
At initialization, an empty set S of selected variables is created, all variables V are considered for inclusion in S ($R \leftarrow V$, where R is the set of variable considered for inclusion) and each variable is considered equivalent only to itself ($Q_i \leftarrow i$). During the main loop the algorithm alternatively attempts to (a) include in S the variable maximally associated with T conditioned on any possible subset of the variables already selected and (b) exclude from S any variable X that is not any more associated with T given any subset Z of other variables in S. Once a variable X is excluded from S, it cannot be inserted any more.
However, before eliminating X from S, the algorithm tries to identify any variable Y in Z that is equivalent to X, by verifying whether $p_{YT.Z'} > a$ when $Z' \leftarrow (Z \cup \{X\}) \setminus \{Y\}$. If such a variable exists, the list of X-equivalent variables Q_X is added to Q_Y (in contrast, the iTIE* algorithm tests whether $p_{ZT.Y} > a$, i.e., it checks if the whole set Z is equivalent to Y). Finally, all equivalence lists Q_i, $i \in S$, are returned as output.
One can then build a predictive signature by choosing *one and only one variable* from each variable set Q_i. To fix the ideas, let's assume that **E** contains three queues, namely $Q_1 = \{X_1, X_4\}$, $Q_3 = \{X_3\}$ and $Q_7 = \{X_7, X_2\}$. Then there are a total of $2 \cdot 1 \cdot 2 = 4$ possible signatures, i.e., $S_a = \{X_1, X_3, X_7\}$, $S_b = \{X_1, X_3, X_2\}$, $S_c = \{X_4, X_3, X_7\}$, $S_d = \{X_4, X_3, X_2\}$. In contrast, the sets $\{X_1, X_2\}$ and $\{X_1, X_4, X_3, X_7\}$ are not valid equivalent signatures, since the first does not select any variable from Q_3 and the latter includes two variables from the same queue (Q_1).

3. Package implementation

The **MXM** package for R currently implements the algorithm SES along with a variety of different conditional independence test. Conditional independence tests $ind(X, T|W)$ are the cornerstones upon which constraint-based algorithms, including SES, are built. Interestingly, constraint-based algorithms can be applied to different types of data as far as they are equipped with a conditional independence test suitable for the data at hand. Similarly, the `SES` function can be applied on different types of outcome (continuous, time-to-event, categorical) and predictors (continuous, categorical, mixed) if the appropriate test is provided. We have implemented a number of these tests in order to grant the user a wide flexibility in terms of the data analysis tasks that can be addressed with the **MXM** package. The `SES` function even allows the users to provide their custom function for assessing conditional independence. The following subsections further illustrate and elaborate upon the implemented functions.

3.1. Conditional independence tests

Assessing dependence among two random variables is one of the oldest problems in statistics, yet it is far from being solved (**??**). Evaluating the conditional independence $ind(X, T|W)$ is further complicated by the presence of the conditioning set W. Moreover, one may desire to deal with cases when X, T and W are all continuous, categorical, or mixed. Two methods often used in the area of constraint-based algorithms are the Fisher's (**?**) and the G^2 tests

Algorithm 1 SES

Input:
Data set on n predictive variables V
Target variable T
Significance threshold a
Max conditioning set k

Output:
A set E of size n of variables sets Q_i, $i = 1, \ldots, n$ such that one can construct
a signature by selecting one and only one variable from each set Q_i

//Remaining variables
$R \leftarrow V$
//Currently selected variables
$S \leftarrow \oslash$
//Sets of equivalences
$Q_i \leftarrow i$, for $i = 1, \ldots, n$

while $R \neq \oslash$ **do**
    for all $X \epsilon \{R \cup S\}$ **do**
        if $\exists Z \subseteq S \setminus \{X\}, |Z| \leq k, s.t., p_{XT.Z} > a$ **then**
            $R \leftarrow R \setminus \{X\}$
            $S \leftarrow S \setminus \{X\}$

            //Identify statistical equivalences, i.e., X and Y seem interchangeable
            if $\exists Y \epsilon Z, s.t., Z' \leftarrow (Z \cup \{X\}) \setminus \{Y\}, p_{YT.Z'} > a$ **then**
                $Q_Y \leftarrow Q_Y \cup Q_X$

            end ifendif

        end ifendif

    end forendfor

    $M = argmax_{\{X \epsilon R\}} min_{\{Z \subseteq S, [Z] \leq k\}} - p_{XT.Z}$
    $R \leftarrow R \setminus \{M\}$
    $S \leftarrow S \cup \{M\}$

end whileendwhile

Repeat the for-loop one last time
//Pack all the identified equivalences in one data structure
$E \leftarrow \oslash$
for all $i \epsilon S$ **do**
    $E \leftarrow E \cup \{Q_i\}$

end forendfor

return E

(**?**). The former is based on partial correlations and assumes continuous measurements and multivariate normality, while the latter is based on contingency tables and can be applied on categorical data. Both tests are implemented in **MXM** in the functions `testIndFisher` and `gSquare`, respectively.
Beside these two functions, we have devised and implemented a number of different conditional independence tests following an approach presented by **?**. Briefly, $ind(X, T|W)$ can be assessed by comparing two nested-models, mod_0 and mod, obtained by regressing the target variable T on the conditioning set W alone and on the conditioning set along with the candidate variable X, respectively. In R language formulas, $mod_0 = T \sim W$ and $mod = T \sim X + W$. The p value $p_{XT.W}$ can be computed through a log-likelihood ratio or χ^2 test, depending on the nature of the two models. Table **??** summarizes the conditional independence tests implemented in **MXM**. Each test is characterized by (a) the type of outcome and predictors it can deal with and (b) the regression method used (if the test is derived according to the approach from **?**). Some of the tests have the option of employing a robust version of the original regression method.

3.2. SES implementation

The `SES` function has been implemented with the aim of making its usage as intuitive as possible for the user. Only two inputs are required, the matrix of predictor variables `dataset` and the outcome variable `target`. The first can be either a numeric matrix, a data frame or an object of the class `ExpressionSet` from the Bioconductor package **affy** (**?**). The outcome can be encoded either as a numerical vector, a (ordered) factor, or an object of the `Surv` class defined in package **survival** (**?**).
Depending on the `dataset` and `target` specified by the user, the `SES` function is able to automatically select the data analysis task to perform and the conditional independence test to use:

1. Binary classification: in a binary classification task the objective of the analysis is to find the model that better discriminates between two classes. An example of binary classification is discerning among Alzheimer and healthy patients on the basis of clinical data. If the `target` variable is a factor with two levels, the `SES` function automatically assumes that the problem is a binary classification task. The default conditional independence test used is `testIndLogistic`.

2. Multi-class classification: this tasks is similar to the binary classification task, but more than two classes are present. These classes may have an intrinsic order, e.g., they represent progressively more severe stages of the same cancer, or they may be independent by each other, as for totally different types of diseases. In the first case an ordered factor should be provided as `target` variable, while a non-ordered factor should be provided in the second case. In both cases the default conditional independence test is `testIndLogistic`, which automatically switches between multinomial logistic (nominal outcome) or ordered logit (ordinal outcome) regression (**?**).

3. Regression: In this case the scope of the analysis is to predict the values of a continuous target, for example the expression of a given gene. For regression tasks the `target` variable should be encoded as a numeric vector, and depending whether `dataset` contains

Name	Outcome	Predictors	Regression	Robust option
`testIndFisher`	Continuous	Continuous	Linear regression	Yes
`testIndSpearman`	Continuous	Continuous	Linear regression	No
`gSquare`	Categorical	Categorical	Contingency tables	No
`testIndReg`	Continuous	Mixed	Linear regression	Yes
`testIndRQ`	Continuous	Mixed	Quantile regression	No
`testIndBeta`	Proportions	Mixed	Beta regression	No
`testIndPois`	Count variable	Mixed	Poisson regression	No
`testIndNB`	Overdispersed count variable	Mixed	Negative binomial regression	No
`testIndZIP`	Zero inflated count data	Mixed	Zero inflated poisson regression	No
`censIndCR`	Survival outcome	Mixed	Cox regression	No
`censIndWR`	Survival outcome	Mixed	Weibull regression	No
`testIndClogit`	Case-control	Mixed	Conditional logistic regression	No
`testIndLogistic`	Categorical	Mixed	Logistic regression	No
`testIndLogistic`	Categorical	Mixed	Logistic regression	No
`testIndSpeedglm`	Continuous, binary or counts	Mixed	Linear, logistic and poisson regression	No

Table 1: Conditional independence tests implemented in **MXM**. For each test the type of outcome, predictors, and regression method is specified in the respective columns. Some of the tests can also employ a robust version of their respective regression method.

solely continuous or mixed (categorical/continuous) predictors the `SES` function uses the `testIndFisher` or the `testIndReg` as conditional independence test, respectively.

4. Time-to-event / Survival analysis: the scope of this type of analysis is to estimate the incidence of an event over time. Survival analysis is conceptually similar to regression, but differs for the presence of censorship, i.e., the exact time-to-event may be unknown for part of the samples. Time-to-event analysis requires a `Surv` object (package **survival**) as `target` variable, and the default conditional independence test is the `testIndCR`.

The user can override the default behavior of the SES function by directly specifying a test to use or by providing a custom function for assessing conditional independence. For example, the user can decide to use the testIndPois instead of the testIndFisher if target contains count values. The user can furthermore control the operation of the SES algorithm by specifying the values for the hyper-parameters a and k. The signature of the method along with a short explanation of its arguments now follows:

```
R> SES(target, dataset, max_k = 3, threshold = 0.05, test = NULL,
+    user_test = NULL, hash = FALSE, hashObject = NULL, robust = FALSE,
+    ncores = 1)
```

- target: the class variable, encoded as a vector, factor, an ordered factor or a Surv object. If a character or an integer is provided, then the corresponding column in dataset is used as target.
- dataset: either a data frame or a matrix (columns = variables , rows = samples). Alternatively, an ExpressionSet object from the package **BioBase** (**?**).
- max_k: the maximum size for the conditioning set to use in the conditional indepedence test.
- threshold: cut-off value for assessing p values significance.
- test: the conditional independence test to use. If NULL, the SES function automatically determines a suitable test depending on target and dataset.
- user_test: a user-defined conditional independence test (provided as a closure type object). If user_test is provided, the test argument is ignored.
- hash: logical variable which indicates whether to store (TRUE) or not (FALSE) the statistics calculated during SES execution in a hash-type object. Default value is FALSE. If TRUE the hash Object is produced and returned in the SES output.
- hashObject: a list with the hash objects generated in a previous run of SES. Each time SES runs with hash=TRUE it produces a list of hash objects that can be re-used in order to speed up next runs of SES.
- robust: A boolean variable which indicates whether (TRUE) or not (FALSE) to use a robust version of the statistical test if it is available. It takes more time than a non-robust version but it is suggested in case of outliers. Default value is FALSE.
- ncores: An integer value indicating the number of CPUs to be used in parallel during the first step of the SES algorithm, where univariate associations are examined.

Internally, the SES function has been optimized in order to improve computational performances. Particularly, the code has been optimized at three different levels:

- Algorithmic level: constraint-based algorithms' computational time is mainly spent for assessing conditional independence. We adopted an algorithmic optimization already presented in **?** in order to avoid performing twice the same conditional independence

test. Assuming variable Y enters in S at iteration n, so that $S_{n+1} = S_n \cup Y$, then the minimum association (maximum p value) between any eligible variable X and the target T conditioned on any subset of S_{n+1} can be written as:

$$\max\left(\max_{Z\subset S_n\setminus X} p_{XT.Z}, \max_{Z\subset S_n\setminus X} p_{XT.Z\cup Y}\right)$$

That means that at each iteration only the conditioning sets including the new variable Y should be taken in consideration for assessing $p_{XT.Z}$, if the quantity $\max_{Z\subset S_n\setminus X} p_{XT.Z}$ has been previously stored.

- Caching intermediate results: The `SES` function can re-use the results of conditional independence tests calculated in a previous run in order to speed-up successive computations. This feature is specifically devised for cases when the method must be run on the same data with different configuration of the hyper-parameters a and k.

- Parallel computing: the first step of the SES algorithm separately assesses the univariate association of each variable with the target T; this is a prototypical example of embarrassingly parallel task, that can be executed on multiple CPUs, by setting the `ncores` argument of the `SES` function equal to 2 or more.

4. Using `SES`

In this section, we provide examples of the use of the `SES` function on simulated, continuous data. All examples were run with **MXM** version 0.8.

4.1. Installing and loading the package

MXM and its dependencies are available from the Comprehensive R Archive Network (CRAN). The package does not require any external dependency for data analyses tasks that can be addressed with the `testIndFisher` conditional independence test (i.e., both predictors and outcome are continuous variables). A number of external packages are required for using the other conditional independence tests: **ordinal** (**?**) and **nnet** (**?**) for the `testIndLogistic` with ordinal and multinomial outcome, respectively. Package **survival** is needed for the `censIndCR`, `censIndWR` and `testIndClogit`, while **quantreg** (**?**) and **betareg** (**?**) are necessary for the `testIndRQ` and `testIndBeta` respectively. Test `testIndZIP` is based on package **pscl** (**?**). Package **MASS** (**?**) is required for performing some of the log-likelihood ratio tests, for the robust version of the Fisher's test and for `testIndNB`. Packages **gRbase**[2] (**?**) and **hash** (**?**) are suggested for faster computations, while **foreach** (**?**) and **doParallel** (**?**) allow for parallel computing during the first step of the algorithm. Finally, `SES` supports `ExpressionSet` objects as input if the Bioconductor package **Biobase** is present and loaded.

```
R> install.packages("MXM")
R> library("MXM")
```

[2]Some dependencies of **gRbase** package are not available on CRAN, however the users can install them directly from the Bioconductor repository.

4.2. Discovering multiple feature signatures

In the following example we simulate a simple continuous data set where the target variable is associated with a subset of the predictors. Collinear variables are then included in the data set in order to create equivalent signatures. `SES` is then run with fixed threshold a and maximum conditioning set k. Successively, we re-run the `SES` function with a different configuration on the same data, but this time we re-use the p values previously computed and stored as a hash object. The results show both the capability of `SES` in retrieving the correct equivalent signatures and the gain in computational time ensured by the hash-based mechanism.

First run of `SES` on simulate data:

```
R> set.seed(12345678)
R> install.packages("hash")
R> require(hash)
R> dataset <- matrix(runif(1000 * 300, 1, 100), nrow = 1000, ncol = 300)
R> target = 3 * dataset[ , 10] + 2 * dataset[ , 200] +
+    3 * dataset[, 20] + runif(1000, 0, 10)
R> dataset[ , 15] <- dataset[ , 10]
R> dataset[ , 250] <- dataset[ , 200]
R> dataset[ , 230] <- dataset[ , 200]
R> system.time(sesObject <- SES(target, dataset, max_k = 5,
+    threshold = 0.2, test = "testIndFisher", hash = TRUE,
+    hashObject = NULL))
```

The output of the `SES` function is an object of the class `SESoutput` with fields:

- `selectedVars`: The selected variables, i.e., the signature of the target variable.
- `selectedVarsOrder`: The order of the selected variables according to increasing p values.
- `queues`: A list containing lists (queues) of equivalent features, one for each variable included in `selectedVars`. A signature equivalent to `selectedVars` can be built by selecting a single feature from each queue.
- `signatures`: A matrix reporting all equivalent signatures (one signature for each row).
- `hashObject`: The `hashObject` caching the statistic calculated in the current run.
- `pvalues`: This vector reports the strength of the association of each predictor with the target, in the context of all other variables. Specifically, for each variable X the maximal p value found over all $ind(X, T|\mathbf{Z})$ executed during the algorithm is reported. Lower values indicate higher association.
- `stats`: the statistics corresponding to the reported `pvalues`.
- `max_k`: The `max_k` option used in the current run.
- `threshold`: The threshold option used in the current run.

- `runtime`: The run time of the algorithm. A numeric vector. The first element is the user time, the second element is the system time and the third element is the elapsed time.
- `test`: The name of the statistical test used in the current run.
- `rob`: The value of the robust option, either TRUE or FALSE.

Generic functions implemented for the `SESoutput` object are:

- `summary(x = SESoutput)`: Summary view of the `SESoutput` object.
- `plot(object = SESoutput, mode = "all")`: Bar plots of the p values for the current `SESoutput` object in comparison to the threshold. Argument `mode` can be either "all" or "partial", using only the first 500 p values of the object.

```
R> summary(sesObject)

Selected Variables: [1]  10  20 200

Selected Variables ordered by pvalue: [1]  10  20 200

Queues' summary (# of equivalences for each selectedVar):

                 10 20 200
#of equivalences  2  1   3

Number of signatures: [1] 6

hashObject summary:
            Length Class Mode
stat_hash   180    hash  S4
pvalue_hash 180    hash  S4

Summary of the generated pvalues matrix:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.3115  0.5062  0.5142  0.7223  1.0000

Summary of the generated stats matrix:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.2237  0.5300  0.5626  0.8646  1.2740

max_k option: [1] 5

threshold option: [1] 0.2

Test: testIndFisher
Total Runtime:
```

```
   user  system elapsed
   0.12    0.00    0.12

Robust:
[1] FALSE
```

Variable 20 *must* be included in the final model. The user can then choose one predictor between variable 10 and another, and one between variable 200 and another two. The resulting six equivalent model have approximately the same predictive performance and are all based on three predictors.

We now re-apply the `SES` function *on the same data* by using the cached statistics used in the previous run. The results are identical, and the computational time significantly decreases.

```
R> hashObj <- sesObject@hashObject
R> sesObject2 <- SES(target, dataset, max_k = 2, threshold = 0.01,
+    test = "testIndFisher", hash = TRUE, hashObject = hashObj)
R> summary(sesObject2)

Selected Variables: [1]  10  20 200

Selected Variables ordered by pvalue: [1]  10  20 200

Queues' summary (# of equivalences for each selectedVar):
                 10 20 200
#of equivalences  2  1   3

Number of signatures: [1] 6

hashObject summary:
            Length Class Mode
stat_hash   180    hash  S4
pvalue_hash 180    hash  S4

Summary of the generated pvalues matrix:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.2399  0.4598  0.4780  0.6985  1.0000

Summary of the generated stats matrix:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.3815  0.7222  0.7960  1.1600  2.3590

max_k option: [1] 2

threshold option: [1] 0.01

Total Runtime:
```

```
   user  system elapsed
   0.01    0.00    0.01

Robust:
[1] FALSE
```

4.3. Identifying the best combination of SES hyper-parameter

Selecting the best configuration of hyper-parameters is an important step in any data analysis task; finely tuning a statistical method often allows to achieve significantly better performances than naively using the default configuration. The package **MXM** provides a dedicate function, namely `cv.ses`, for automatically identify the optimal configuration for the SES algorithm hyper-parameters a and k. This function internally applies a model selection schema based on stratified cross-validation (**?**).

More in detail, `cv.ses` partitions the available data in a given number of folds, each containing approximately the same number of samples. Each fold is used in turn as test set, while the remaining data form the training set. The latter is used for training a predictive model on the features selected by `SES`, model that is successively applied on the test set for obtaining testable predictions. Performances are computed for each fold and then averaged. This whole procedure is repeated for each combination of a and k over their respective, user-specified ranges, and the optimal configuration $\{a^*, k^*\}$ correspond to the values that produced the best average performances. The users can either provide their own pre-specified folds, or have them generated internally within `cv.ses` by the function `generateCVRuns` of the package **TunePareto** (**?**).

The type of predictive model to fit on the training data, as well as the performance metric to use depends on the data analysis task at hand. For classification tasks, logistic regression and the receiver operator characteristic (ROC) area under the curve (AUC, **?**) are the default choice. The AUC is computed with the **ROCR** (**?**). Regression problems are addressed with standard linear regression and the mean square error (MSE) metric, the latter defined as $\sum_i (y_i - \hat{y}_i)^2/n$, where n is the number of test instances and y_i, $\hat{y}_i$ are the actual target value and the prediction for instance i, respectively. Survival analysis tasks require specialized methods, namely the Cox proportional-hazards model (**?**) and the concordance index (CI, **?**) performance metric. The CI has an interpretation similar to the AUC, ranging in $[0, 1]$ with 0.5 indicating random predictions and 1 corresponding to a perfect rank of the test instances. Package **Hmisc** (**?**) is required for the computation of the CI metric. The user has also the possibility of providing customized functions for predictive modeling and performance evaluation. The signature of the `cv.ses` function is the following:

```
R> cv.ses(target, dataset, kfolds = 10, folds = NULL, alphas = NULL,
+    max_ks = NULL, task = NULL, metric = NULL, modeler = NULL,
+    ses_test = NULL)
```

The argument `target` and `dataset` are as in the `SES` function. Other arguments are specified below.

- `kFolds`: The number of folds to partition the data in.

- `folds`: A list specifying the folds to use. If provided than `kFolds` is ignored.
- `alphas` and `max_ks`: The ranges of values to be evaluated for the hyper-parameters a and k, respectively.
- `task`: A character specifying the type of task to perform:"C" for classification, "R" for regression and "S" for survival analysis.
- `metric`, `modeler`, `ses_test`: user-specified functions for the performance metric, predictive modeling and conditional independence test to use, respectively.

We now apply the `cv.ses` function to the simulated data presented in Section **??**:

```
R> cv.ses.object = cv.ses(target, dataset, kfolds = 10, task = "R")
```

The best SES configuration and its respective performance can be easily retrieved:

```
R> cv.ses.object$best_performance

[1] -8.794793

R> cv.ses.object$best_configuration

$id
[1] 2

$a
[1] 0.1

$max_k
[1] 2
```

5. Experimental validation

We further evaluate the capabilities of the SES algorithm and **MXM** package on real data. Particularly, we aim at investigating (a) if the signatures retrieved by the algorithm provide statistically-equivalent predictive performances, and (b) whether these performances are comparable with the ones provided by the state-of-the-art feature selection algorithm LASSO, as implemented in the R package **glmnet** (**?**). All data and scripts for replicating the results of this comparison are freely available as supplementary material.

5.1. Data sets description

We use three different data sets for our experiments. All data sets are formed by continuous predictors, but largely differ in the number of samples / variables and in the type of outcome (see Table **??**). Moreover, each data set comes from a different application domain. The

first, Breast Cancer, is targeted at the discrimination of estrogen-receptor positive (ER+) or estrogen-receptor negative (ER-) tumors using gene expression measures. This data set is publicly available in the package **breastCancerVDX** (**?**). The AcquaticTox data set leads to a Quantitative Structure-Activity Relationship (QSAR) regression problem. Data are freely available from the package **QSARdata** (**?**). The task here is to predict the toxicity of 322 different compounds on the basis of a set of 6652 molecular descriptors produced by the software DRAGON (Talete Srl, Milano Italy). The Vijver-2002 data (**?**) contains the expression measures of breast cancer patients and the aim is to relate them with their survival time.

Name	# samples	# variables	Task	Outcome
Breast Cancer	17816	286	Classification analysis	Binary, rarest class frequency: 36%
AquaticTox	322	6652	Regression analysis	Continuous
Vijver-2002	295	70	Survival analysis	Right-censored, number of events: 88

Table 2: Data sets used for the experiments. For each data set the table reports the number of samples, the number of variables/predictors, task to accomplish (classification/regression/survival analysis) and the type of outcome. References for each data set are reported in the text.

5.2. Experimentation protocol

Derivation and assessment of predictive models

In order to empirically evaluate the performance of the proposed method we have repeated the following experimentation procedure 500 times, each time using different data splits.
First, data are split in a training set D_{train} and in a hold-out set $D_{holdout}$, each containing 50% of all samples. The best hyper-parameter configuration for `SES` is identified on the training set through a ten-fold cross-validation model selection procedure, where the `SES` hyper-parameters are varied within $a \in [0.01, 0.05, 0.1]$ and $k \in [3, 5]$. `SES` is then run on the whole D_{train} with the best hyper-parameters for identifying the optimal signatures. A predictive model for each signature is finally estimated based on D_{train} and applied on $D_{holdout}$ for estimation of the performance. Logistic, linear and Cox regression procedures are used for obtaining the predictive models, depending on whether the outcome is binary, continuous or time-to-event, respectively. Appropriate performance metrics are used accordingly: AUC is used for binary classification, continuous outcomes are evaluated through the MSE, while CI is used for evaluating the performance of Cox regression models. MSE quantifies the prediction error, thus lower values indicate better performances, while the reverse holds for AUC and CI. Data splitting is stratified for classification and survival tasks, i.e., an equal proportion of

	Number of signatures									
	1	2	3	4	5	6	7	8	9	10+
Breast Cancer	301	136	7	40	1	2	0	3	1	9
AquaticTox	17	17	15	12	9	16	4	18	13	379
Vijver-2002	31	181	13	94	3	55	1	37	3	82

Table 3: Frequency of signature multiplicity. Each cell reports how many times j equivalent signatures are retrieved for its corresponding data set (row), where j is the number reported on top of the cell's column. The notation 10+ indicates 10 or more signatures.

instances of each class (or of censored/non-censored cases) is kept in each data split.

5.3. Contrasting against LASSO

SES and LASSO are used in turn as feature selection algorithm in the experimentation protocol described in Section **??**, and they are compared on the basis of the performances obtained on all $D_{holdout}$. In each repetition the same data split is used for both algorithms, in order to ensure a fair comparison. Recall that LASSO selects only a single subset of relevant variables, while SES potentially retrieves multiple signatures. Thus we arbitrarily select the first signature retrieved by SES for comparison with LASSO. This is not necessarily the best one, and can be deemed to be chosen with a systematic random sampling. In the cross-validation step of the experimentation protocol, the range of values over which we optimize the hyper-parameter λ for the LASSO algorithm is automatically determined by the least angle regression (LARS, **?**) fitting procedure.

5.4. Results

Assessing the equivalence of SES signatures

Table **??** reports the distribution over 500 repetitions of the number of signatures identified by SES for each dataset. Each row refers to one dataset, while each column to a given number of signatures. The results indicate that the number of retrieved signature is highly dependent upon the specific dataset. Particularly, both AquaticTox and Vijver-2002 tend to produce a large number of equivalent signatures, while a single signature is found for the Breast Cancer dataset 301 times in 500 repetitions. Interestingly, the number of retrieved signatures is highly variable across repetitions: for the AquaticTox dataset, simply splitting the data in different D_{train} and $D_{holdout}$ sets lets the number of signatures range from 1 to 292032. This shows that the detection of equivalent features is strongly influenced by the specific sample at hand.

We now investigate whether the retrieved signatures achieve performances that are actually equivalent. For each data set and for each repetition where at least two signatures are retrieved, we compute the SES performances' coefficient of variation (CV). The CV is defined as the ratio between standard deviation and mean value, and it measures the dispersion of a

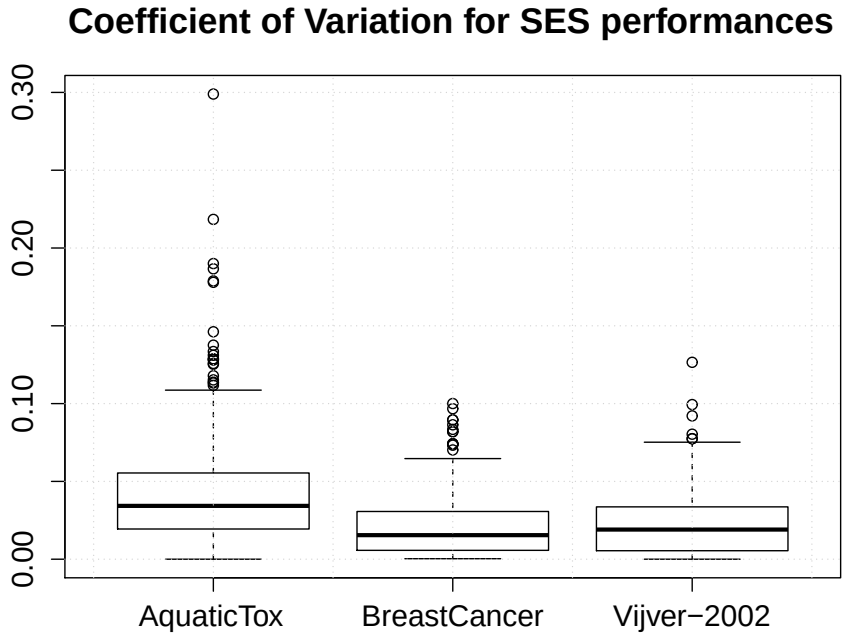

Figure 1: Boxplot of the SES performances' coefficient of variation across the 500 iterations for each dataset.

distribution standardized with respect to the magnitude of its measurements. Figure **??** and Table **??** show that in all data sets the median CV value is well below 5%, indicating that within each repetition the performances of the retrieved signatures are extremely close to each other. The AquaticTox data set produces the highest CV values, marked as circles/outliers in Figure **??** (two extreme CV values, reaching 1.19 and 0.97 respectively, were removed for the sake of figure readability). We also observe that the higher the number of signatures, the higher the coefficient of variation (Spearman correlation: 0.69 Vijver, 0.25 Breast Cancer, 0.45 AquaticTox data set, p value < 0.001 in all cases). This result is not unexpected. When few signatures are retrieved, each signature differs from the other for only one or two features, and thus their predictive performances are expected to be similar. When thousands of signatures are produced, their heterogeneity increases, as well as the deviation of their performances. It can be concluded though that the algorithm is generally stable, with very rare exceptions, and leads in general to signatures with very close predictive performance. It could be argued that the variation in the estimated predictive performance is often an order of magnitude lower than the performance estimates themselves.

Contrasting SES and LASSO

Table **??** shows the 95% confidence intervals of the paired differences in performance between SES and LASSO, computed over 500 repetitions. For each data set, the differences are computed in such a way that positive values indicate SES outperforming LASSO, and vice versa. The table shows that for both the Vijver-2002 and AquaticTox data sets the confidence inter-

vals cross zero, thus on these two data sets the SES and LASSO methods are not statistically different at 0.05 significance level. LASSO performs slightly better in the Breast Cancer data set though. Figure **??** reports the distribution for the differences in performances between the two methods. Here the equivalence between the two methods on the Vijver-2002 is even more evident, while differences in performances for the AquaticTox dataset show quite a large variability.
Table **??** shows the distribution of the number of selected variables over the 500 repetitions. SES is generally quite parsimonious, and it usually selects the same number of variables, independently by the data split, as demonstrated by the low standard deviations over the 500 repetitions. In contrast, the number of variables selected by LASSO widely varies across repetitions. SES also selects much fewer variables than LASSO for both the AquaticTox and Vijver-2002 data sets, while for the Breast Cancer dataset LASSO produces only slightly more parsimonious models but again with larger variability.

6. Discussion and conclusions

In the present work we introduced the R package **MXM**, which implements the SES algorithm for selecting statistically-equivalent sets of predictive signatures. The package further implements a number of conditional independence tests able to cope with a variety of data types. These tests can be used alone (for inference or causal discovery) or in conjunction with the `SES` function, in order to deal with several data-analysis tasks, including (multi-class) classification, regression and survival analysis.
We used three real-world, publicly available data sets from different application areas for evaluating the capabilities of the software. Multiple signatures were actually identified for all data sets, indicating that equivalences among predictors are frequently present in real applications. Deviation among the signatures' performances proved to be particularly low, indicating that the signatures have almost equal predictive power.
We further contrasted the performance of the SES algorithm against the LASSO method. We attempted to have a comparison as fair as possible, so we always compared the LASSO signature against the first one retrieved by SES. In the context of our experiments, SES was more stable in terms of number of variables selected across different data splits, while LASSO in general selects a higher number of variables. The two methods had quite comparable performance, with LASSO performing slightly better in the Breast Cancer example. These results are in agreement with previous theoretical results (**?**): under some quite general conditions,

	2.5%	Median	97.5%
Breast Cancer	0.06%	2.17%	8.36%
AquaticTox	0.06%	4.20%	12.80%
Vijver-2002	0.04%	2.18%	6.44%

Table 4: Quantiles of the coefficient of variation (CV) of the SES performances. Results are reported separately for each data set (rows).

	2.5%	Mean	97.5%
Breast Cancer	-0.222833	-0.104123	-0.001349
AquaticTox	-0.228774	-0.115798	0.027270
Vijver-2002	-0.096756	-0.016740	0.033616

Table 5: Quantiles of the difference in performance between SES and LASSO. Positive values indicate SES outperforming LASSO.

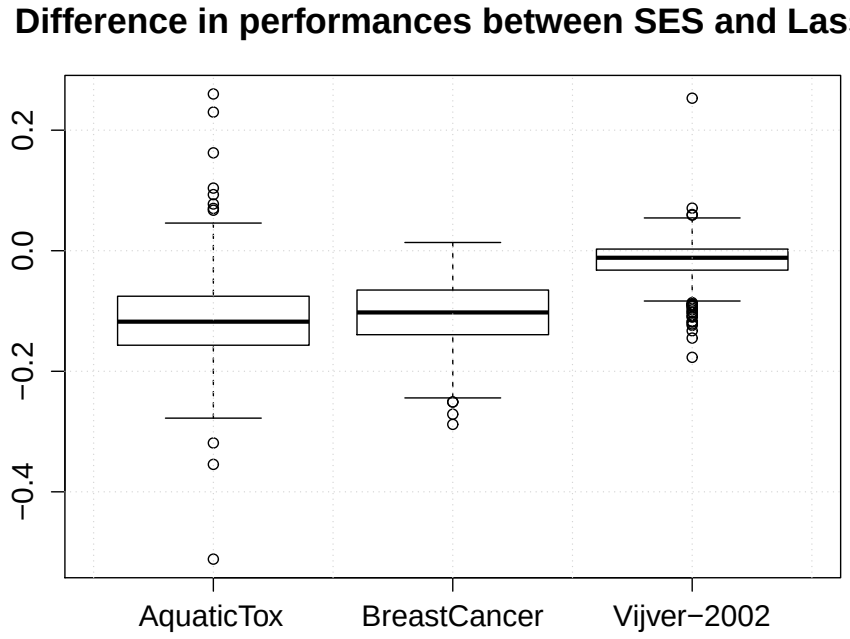

Figure 2: Boxplot of the difference among SES and LASSO performances across the 500 iterations for each dataset.

LASSO retrieves a super set of the Markov-Blanket of the target variable, meaning that all variables needed for optimal prediction plus some noisy variables are selected. In contrast, SES is devised for retrieving the Parent-Children set of the target variable, i.e., a subset of the Markov-Blanket. Thus, it is not surprising that in our experimentation SES selects fewer variables and does not outperform LASSO. We also note that these results may be influenced by the restricted range of values over which SES hyper-parameters a and k have been optimized.

The aim of this paper is not an assessment of SES, of course; and results in Section **??** are not conclusive. A more extensive comparison study is currently under preparation in order to exhaustively evaluate SES capabilities and contrast its performance against a range of feature

	Average SES	Average LASSO	StD. SES	StD. LASSO
Breast Cancer	13.29	10.62	5.91	15.43
AquaticTox	5.68	160.75	1.80	66.34
Vijver-2002	3.24	10.50	0.98	3.64

Table 6: Distribution of the number of variables selected by SES and LASSO. For each method and data set both the average number and the standard deviation (St.D.) of selected variables is reported.

selection methods.

In conclusion, our limited experiments indicate that:

- Multiple, equally-performing signatures naturally occur in real-world data sets, either due to equivalence among predictors or to impossibility to distinguish them due to limited sample size. In either case, this phenomenon should be duly taken into account while retrieving predictive feature subsets.
- The signatures retrieved by the SES algorithm provide predictive performances extremely close to each other in all data sets included in our analyses, demonstrating in fact to be equally-predictive.
- SES and LASSO provide comparable results, and SES is generally more parsimonious and sheds light on the characteristics of the problem at hand by identifying equivalences hidden into the data.

We keep developing **MXM** by adding new conditional independence tests, as well as new functionalities. For example. the MMPC algorithm, which performs feature selection without providing multiple solutions, and the PC and MMHC algorithms, two methods for constructing the skeleton of a Bayesian network. Future work will focus on both algorithmic and implementation improvements. In future extensions SES will attempt to retrieve the Markov Blanket of the target variable, i.e., the variables set with theoretically the highest predictive power. The aggregation of models trained on equivalent signatures for improving predictive performances is also under consideration. In addition, we aim at extending **MXM** in the areas of model selection and performance estimation (**?**), two fields closely related to the problem of feature selection.

7. Acknowledgments

The work was co-funded by the STATegra EU FP7 project, No 306000, by the EPILOGEAS GSRT ARISTEIA II project, No 3446, and by the European Research Council (ERC) project No 617393, "CAUSALPATH - Next Generation Causal Analysis". We sincerely thank Damjan Krstajic and Giorgos Borboudakis for their invaluable comments, suggestions and critical reading of the manuscript.

Affiliation:

Ioannis Tsamardinos
Computer Science Department
University of Crete
Voutes Campus
GR-70013 Heraklion, Crete, Greece
Telephone: +30 2810 39 3575
Fax: +30 2810 39 1428
E-mail: tsamard@csd.uoc.gr
URL: http://www.mensxmachina.org/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
September 2017, Volume 80, Issue 7 *Submitted:* 2015-10-01
doi:10.18637/jss.v000.i00 *Accepted:* 2017-09-05