

SISSO

Sure Independence Screening and Sparsifying Operator

(SISSO.3.3)

User Guide

用户指南

Runhai Ouyang

Shanghai

July, 2023

Contents

1. Introduction	1
2. Installation.....	2
3. Running SISSO.....	2
4. Input and output files.....	3
4.1 Input files.....	3
SISSO.in	3
train.dat	3
4.2 Output files.....	4
SISSO.out.....	4
models	5
desc_dat	5
SIS_subspaces	5
convexnD_hull	5
log	6
VS_results.....	6
5. Control parameters	6
5.1 SISSO.in	6
ptype	6
ntask	6
task_weighting	6
scmt	7
desc_dim.....	7
nsample	7
restart.....	7
nsf	7
ops	7
fcomplexity	8
funit	8
fmax_min	9
fmax_max	9

nf_sis.....	9
method_so	10
nl1l0	10
fit_intercept	10
metric	10
nmodels	11
isconvex	11
bwidth.....	11
5.2 VarSelect_SISSO.py.....	11
6. Algorithms	11
6.1 Regression.....	11
6.2 Classification	13
6.3 Variable selection assisted symbolic regression	15
6.4 Multi-task learning	16
6.5 Sign-constrained multi-task learning	17
7. Utilities	18

1. Introduction

SISSO is a data-driven approach that combines symbolic regression and compressed sensing to identify models and descriptors in explicit mathematical expressions for materials science, which was originally proposed in 2018 [Phys. Rev. Mater. 2, 083802 (2018)]. Several implementations of the SISSO method can be available, and this user guide is for the SISSO FORTRAN code created and maintained by Runhai Ouyang at <https://github.com/rouyang2017/SISSO>. Capabilities of the current SISSO code include:

- Regression & Classification
[R. Ouyang, S. Curtarolo, E. Ahmetcik, M. Scheffler, L. M. Ghiringhelli, Phys. Rev. Mater. 2, 083802 (2018)]
- Multi-task learning
[R. Ouyang, E. Ahmetcik, C. Carbogno, M. Scheffler, L. M. Ghiringhelli, J. Phys.: Mater. 2, 024002 (2019)]
- Variable selection assisted symbolic regression
[Z. Guo, S. Hu, Z.-K. Han, R. Ouyang, J. Chem. Theory Comput. 18, 4945 (2022)]
- Sign-constrained multi-task learning
[J. Wang, H. Xie, Y. Wang, R. Ouyang, J. Am. Chem. Soc. 145, 11457 (2023)]

This SISSO code is licensed under the Apache License, Version 2.0. For more details about the terms and conditions for use, reproduction, and distribution, please refer to:

<http://www.apache.org/licenses/>

Permissions:

- ✓ Commercial use
- ✓ Modification
- ✓ Distribution
- ✓ Patent use
- ✓ Private use

Limitations:

- ✗ Trademark use
- ✗ Liability
- ✗ Warranty

Conditions:

- ⦿ License and copyright notice
- ⦿ State changes

2. Installation

A FORTRAN MPI compiler is required to compile the SISO parallel program. Below are two options for compiling the program using an Intel MPI compiler (other compilers may work as well but the Intel compilers are recommended). Download and unzip the package in Linux operating systems, and then go to the folder 'src' and do:

```
1) mpiifort -fp-model precise var_global.f90 libsisso.f90 DI.f90 FC.f90 SISO.f90 -o  
   ~/bin/SISO
```

or

```
2) mpiifort -O2 var_global.f90 libsisso.f90 DI.f90 FC.f90 SISO.f90 -o ~/bin/SISO
```

The option (1) enables better accuracy and run-to-run reproducibility of floating-point calculations; option (2) is about 2X faster than (1) but tiny run-to-run variations may happen between processors of different types, e.g. Intel and AMD.

If 'mpi' related errors present during the compilation, try opening the file 'var_global.f90' and replace the line "use mpi" with "include 'mpif.h'". However, " use mpi " is strongly encouraged (see <https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report/node411.htm>).

Modules of the program:

var_global.f90	! declaring the global variables.
libsisso.f90	! subroutines and functions for mathematical operations in SISO.
DI.f90	! model sparsification (descriptor identification).
FC.f90	! feature construction.
SISO.f90	! the main program.

3. Running SISO

To run SISO for regression, classification, multi-task learning, sign-constrained multi-task learning, the two files, SISO.in and train.dat, have to be present in the working directory. Their templates can be found in the folder "input_templates". The simple command-line usage is:

```
SISO > log
```

You may need to remove resource limit by running the command 'ulimit -s unlimited'. However, users are recommended to run the code on clusters, as the SISO jobs often require many CPU cores and much memory. For example, use this command in your submission script:

```
mpirun -np number_of_cores SISO > log
```

The VS-SISO algorithm enable symbolic regression with large number of input features. The program VarSelect_SISO.py (in the folder “utilities”) was created to do this job. There are calculation parameters to be defined by users in the VarSelect_SISO.py. With the presence of the prepared SISO.in, train.dat, and the VarSelect_SISO.py in the working directory, users can run the VS-SISO with the command in their submission script:

```
Python3 VarSelect_SISO.py
```

4. Input and output files

4.1 Input files

SISO.in

The file SISO.in contains most of the control parameters for users to define the SISO job on the information of the type of machine learning (regression, classification, multi-task learning, sign-constrained multi-task learning), model complexity, number of data and input features, sparsifying operators and so on. The code reads the file SISO.in line by line to detect the keywords and corresponding values. Blank line, space, and comments starting with the exclamation mark (!) will be ignored. Users can find the template for the SISO.in in the folder “input_templates”. Instructions on how to set the control parameters are described in the Section 5.

train.dat

There are several templates available in the folder “input_templates” for the train.dat file according to the specific machine learning job, including the train.dat_regression, train.dat_classification, train.dat_regression_multitask, and train.dat_classification_multitask. That is, users can take the train.dat template corresponding to the SISO job defined in the SISO.in to prepare your training data, and then rename the file as train.dat before running SISO.

In train.dat_regression, the first line are the strings to explain/name each column. Since the second line, each line describes one data, with the first entry being a string to explain this data, e.g. compound name, the second entry being the value of the target property of this compound, and all the rest entries being the values of the input features. Note that users should avoid using the special characters of space, comma, Tab, and the mathematical operators to name a feature.

In `train.dat_classification`, the format is the same as in the `train.dat_regression`, except that the second column for the target property is removed. No property values are needed for classification. SISSO only require the category information of the dataset, which will be provided in `SISSO.in`. For example, supposing there are the three categories A, B, C, with 50, 60, 70 data for them, respectively, you will set “`nsample = (50, 60, 70)`” in the `SISSO.in`. Correspondingly, you need to sequentially arrange the three groups of data in the `train.data`, i.e. the A-50 data from line 2 to 51, the B-60 data from line 52 to line 111, and the C-70 data from line 112 to line 181.

In `train.dat_regression_multitask`, and `train.dat_classification_multitask`, the formats are the same with that of `train.dat_regression` and `train.dat_classification`, respectively, except that the data are sequentially arranged task by task. For example, supposing there are the three tasks T1, T2, T3 in regression, with 100, 200, 300 data for them, respectively, then you will set “`nsample = 100, 200, 300`” in the `SISSO.in`. Correspondingly, you need to arrange the T1-100 data from line 2 to line 101, the T2-200 data from line 102 to line 301, and the T3-300 data from line 302 to line 601. In multi-task classification, supposing there are the two tasks T1 and T2, with the data (50, 60, 70) for the T1 and (80, 90, 70) for T2, then you will set “`nsample = (50, 60, 70), (80, 90, 70)`” in the `SISSO.in`, and arrange the T1-(50, 60, 70) data from line 2 to line 181, and the T2-(80, 90, 70) data from line 182 to line 421.

4.2 Output files

SISSO.out

The information of control parameters, feature spaces, and the identified best model are output to the file `SISSO.out`. For a SISSO-nD calculation (the model dimension is set to n in `SISSO.in`), the models from 1 up to n dimensions will be created. However, the m D models ($1 \leq m < n$) are generated only because they are needed to generate the n D model. These m D models from the SISSO-nD calculation are often not the best as compared from calculations directly for the m D. The reason is that the computationally affordable size of the SIS-subspace for a SISSO- m D job can be far greater than that in the SISSO- n D job, and thus higher probability to get a better model. Therefore, it is recommended to perform a SISSO- x D calculation if you would like to obtain a x D model, with proper SIS-subspace size. It can be possible that when the SIS-subspace size exceeds certain threshold, the model and/or accuracy no longer change with the size, and thus convergence is reached.

For regression, the functional form of a n D model is $y = \sum_1^n \beta_i d_i + \beta_0$, where the $\{d_i\}$ is termed the n D descriptor, $\{\beta_i\}$ are the corresponding coefficients, and β_0 is the intercept. In `SISSO.out`, the `features_ID` after each d_i is the line number of this feature in the file `Uspace.expressions` in the folder `SIS_subspaces`. In multi-task learning regression, all the tasks share the same n D descriptor, but each task have a set of different coefficients.

For classification, there are only descriptors but no coefficients in the `SISSO.out`. A n D

descriptor means a n -dimensional map. Currently, up to 3D are implemented for classification, yet the convex-hull construction for 3D is not stable and bugs remain to be fixed. As described in Section 6.2, the classification error is defined as the number of data in all overlap regions. The overlap size is used as the second metric in model selection. If there is no any overlap between all domains, the overlap size is given with a negative number, whose magnitude indicates the smallest separation distance between the domains. With the identified nD descriptor, one could straightforwardly apply support vector classification to find the separating hyperplanes, or decision tree to find the separating rules.

models

This folder contains the files for the information of the top n models/descriptors, where the n can be set in `SISSO.in`. The features in each model are indicated by their “Feature_ID” which is the line number in the file `Uspace.expressions` in the folder `SIS_subspaces`. For regression, coefficients are also provided in the files `top n _D m _coeff`, in the format of `(c_i,i=0,n)_j,j=1,ntask`, where `c_0` is the intercept, and `c_1` to `c_n` are the coefficients corresponding to `d_1` to `d_n` in the linear model $y = \sum_1^n c_i d_i + c_0$.

desc_dat

The folder `desc_dat` contains the data files of the identified descriptors listed in the `SISSO.out`. The values of `y_true` (from the training data), `y_pred` (from the model prediction), and the descriptor for each compound are provided. The files “`desc_D n _tm.dat`” denotes the data corresponding to the descriptor at dimension n and task m .

SIS_subspaces

There are n SIS-subspaces in a `SISSO- nD` calculation. These subspaces, including the feature expressions and feature data, are saved in the folder `SIS_subspaces`. The file `Uspace.expressions` contains the feature expressions of the union of all these subspaces. Corresponding data of the expressions are available in the files of `Uspace_t n .dat`, where the “ tn ” means the task n . For example, the features in `Uspace.expressions` from line 1 to N have their corresponding data in `Uspace_t001.dat` from column 2 to $N+1$ for regression (the target property occupies the first column), and from 1 to N for classification. The “t001” in the `Uspace_t001.dat` says that these data correspond to the task 1. The features in the `Uspace.expressions` are ranked according to the SIS score, i.e. the correlation between each feature and the target property.

convex nD _hull

In classification, each category domain is approximated by the convex volume of the data

group. The file `convex n D_hull` contains the information to construct the n D volume of a data group in the space of the identified n D descriptor.

log

The log file (`SISSO > log`) has the progress information of a SISSO job. One can monitor the progress and estimate the remaining time of the job.

VS_results

The results of performing VS-SISSO are output to the file `VS_results` which contains the information about the change of models errors, selected variables and other details with the iterations.

5. Control parameters

5.1 SISSO.in

The keywords in `SISSO.in` can be useful for either regression or classification, or both. Those that are only useful for regression are labeled as (R), and those only for classification are labeled as (C). The (R&C) denotes the keywords are useful for both regression and classification.

ptype

ptype = 1 or 2. The type of the target property. 1: regression with continuous properties, 2: classification with categorical properties.

ntask

(R&C). **ntask** = integer. Number of tasks. **ntask** = 1: the usual machine learning with single task; **ntask** > 1: multi-task learning (MTL).

task_weighting

(R). **task_weighting** = 1 or 2. The weighting for MTL regression. 1: no weighting (all tasks treated equally regardless of their respective dataset size); 2: each task is weighted by $N^t / \sum_t N^t$, i.e. the number of data in this task divided by the total number of data in all tasks.

scmt

(R). **scmt** = .true. or .false. Sign-constrained MTL regression is invoked if .true.

desc_dim

(R&C). **desc_dim** = integer. Dimension of the descriptor or model.

nsample

(R) **nsample** = integer, integer, ...

(C) **nsample** = (integer, integer, ...), (integer, integer, ...), ...

Number of samples in the train.dat. For single-task regression, the input is just a single integer. For MTL regression, the input is multiple integers separated by comma for the number of samples of the tasks. For single-task classification, there will be just one round bracket, inside which are the integers separated by comma for the number of data of the corresponding categories. For MTL classification, there will be multiple round brackets separated by comma. Each bracket defines one task, and the integers inside each bracket define the number of data for the categories.

restart

(R&C). **restart** = 0 or 1. 0: starts the job from scratch, 1: continues the job (the progress information of last job was stored in the file CONTINUE)

nsf

(R&C). **nsf** = integer. Number of scalar features provided in the file train.dat. The “scalar feature” means each feature is one column in the dataset.

ops

(R&C). **ops** = '(operator)(operator)(operator)...'. The mathematical operators to be used for feature construction. Users can customize the operators from the list: {+, -, *, /, exp, exp-, ^-1, ^2, ^3, sqrt, cbrt, log, |-, scd, ^6, sin, cos}, where the “exp-”, “cbrt”, “|-, “scd” means exp(-x), cubic root, $|x_1 - x_2|$, and standard Cauchy distribution, respectively.

There are different rungs during the feature construction to generate different tiers of the feature space. rung=0 corresponds to **fcomplexity** =0 (the original space of input features); rung=1 corresponds to **fcomplexity** = 1 (maximal feature complexity, i.e. number of

operators in the feature, is 1); rung=2 corresponds to **fcomplexity** = 2 or 3; and rung=3 corresponds to **fcomplexity** = 4 or 5 or 6 or 7. Higher rungs are allowed but demand huge memory for the calculation. More details are in the introduction to the keyword **fcomplexity**.

The operators can be set to be the same or different for different rungs during feature construction. For example, if rung =3 (**fcomplexity** set to 4 or 5 or 6 or 7), the **ops** = '(+)(-)(*)(/)' means that the operators of +, -, *, / will be used throughout the feature construction of all rungs; and that **ops**= '(+)(-)(*)(/)', '(+)(exp)', '(/)(^2)' means that the operators +, -, *, / will be used for the 1st rung, and the + and exp will be used for the 2nd rung, and the / and ^2 will be used for the 3rd rung.

fcomplexity

(R&C). **fcomplexity** = integer. The feature complexity is defined as the number of operators in a feature. All the input variables and the expressions generated by mathematical combination between the input variables are all termed features. Thus, **fcomplexity**=0 leads to only the input variables in the feature space, and **fcomplexity**=3 will lead to a feature space with all the features having the complexity not greater than 3. Usually, the **fcomplexity** with a number greater than 7 will lead to an extremely huge feature space (rung or tier > 3, see section 6.1) that demands too much memory to be computationally infeasible.

funit

(R&C). **funit**=(integer:integer)(integer:integer)... Each parentheses () denotes one type of unit, and the integers inside the parentheses indicates that the features from n1 to n2 in the file train.dat have the same unit. For example, if in the train.dat the features from the first to the 5th have energy dimension with unit eV, features from the 6th to the 9th have the length dimension with unit angstrom, the 10th feature is dimensionless, and the 11th feature has other different unit, then in SISSO.in one can set **funit** = (1:5)(6:9)(11:11). Features not included in the **funit** are treated as dimensionless. If all input features are dimensionless, then simply comment out this line or remove the keyword **funit**. The purpose is to enable SISSO to perform dimension analysis and avoid unreasonable combinations such as “energy + length” and “energy - length”.

In SISSO, the unit of each feature is represented by a jD vector, where j is the number of unit type (number of parentheses from the input for **funit**). For example, if there are 6 features in train.dat, with their units in eV, eV, eV, Å, Å, Å, which corresponds to funit = (1:3)(4:6) in SISSO.in, then the feature units are represented by

```
feature1: 1  0
feature2: 1  0
feature3: 1  0
feature4: 0  1
feature5: 0  1
```

feature6: 0 1

The linear combinations during feature construction are only allowed between two features with the same unit vector. The unit vector (v) for the new feature generated by mathematical operations on existing features is defined as:

feature1 + feature2 and feature1 – feature2 → new_feature: $v = v1$ (or $v2$)

feature1*feature2 → new_feature: $v = v1 + v2$

feature1/feature2 → new_feature: $v = v1 - v2$

$\log(\text{feature1})$, $\exp(\text{feature1})$, $\sin(\text{feature1})$, $\cos(\text{feature1})$ → new_features: $v = 0$

$(\text{feature1})^n \rightarrow v = v1*n$

SISSO also accept the input of units from a file named “feature_units”. For example, if the units of the 6 input features are in eV, $\text{eV}^{1/2}$, eV, \AA^2 , \AA , \AA , then user can prepare the file in the working directory with the following content:

```
1.0 0.0
0.5 0.0
1.0 0.0
0.0 2.0
0.0 1.0
0.0 1.0
```

Please note that, when using the “feature_units” file, users have to define the **funit** parameter in SISSO.in so that the number of parenthetical arguments is equal to the number of columns in the “feature_units” file.

fmax_min

(R&C). **fmax_min** = float. The threshold that if the maximal absolute value of the data in a feature is smaller than **fmax_min**, it means the magnitudes of all numbers in the feature are so small that the feature will be treated as zero-feature and be discarded.

fmax_max

(R&C). **fmax_max** = float. The threshold that if the maximal absolute value of the data in a feature is greater than **fmax_max**, it means the magnitudes of certain numbers in the feature are so large that the feature will be treated as infinity-feature and be discarded.

nf_sis

(R&C). **nf_sis** = integer (or integer_1, integer_2, ..., interger_desc_dim). Size of the SIS-subspaces. There will be n SIS-subspaces for a SISSO- nD calculation. If the input is just a single integer, then all the n subspaces have the same size. Otherwise, users can input n integers, separated by comma, to specify different size of the subspaces.

method_so

(‘L0’ for R&C, ‘L1L0’ for R only). **method_so** = ‘L0’ or ‘L1L0’. ‘L0’ is the ℓ_0 -norm minimization sparsifying method. ‘L1L0’ is the method proposed by Ghiringhelli et al. [PRL 114, 105503 (2015)], i.e. firstly screening certain number of features, says 30, by using LASSO, then followed with ℓ_0 to identify the best 1D, 2D, 3D, ... models/descriptors. ‘L0’ is recommended for its accuracy over the ‘L1L0’ for low-dimensional models.

nl1l0

(R). **nl1l0** = integer. The number of features selected by LASSO for subsequent ℓ_0 . Other control parameters for the LASSO are not explicitly included in the SISO.in template. Default values for those parameters are:

L1_max_iter=1e6

The maximal number of iterations in LASSO.

L1_dens=120

Density of sampling the penalty parameter λ . The λ_i take values by using the expression: $\log_{10}(\lambda_i) = \log_{10}(\lambda_{\max}) - (i-1)[\log_{10}(\lambda_{\max}) - \log_{10}(\lambda_{\min})]/L1_dens$. Here, the λ_{\max} is the smallest λ for which all the coefficients are zero [J. Friedman, T. Hastie, R. Tibshirani, J. Stat. Softw. 33, (2010)], and the λ_{\min} is defined as $0.001 * \lambda_{\max}$.*

L1_tole=1e-6

The tolerance for the LASSO optimization with a given λ .

L1_minrmse=1e-3

The condition for LASSO to stop trial with new λ if the model LASSO_RMSE < L1_minrmse

L1_warm_start = .true.

Using the solution of last λ to start the next optimization with a new λ .

fit_intercept

(R). **fit_intercept** = .true. or .false. Fit to a nonzero / zero intercept for the linear model.

metric

(R). **metric** = ‘RMSE’ or ‘MaxAE’. The metric for model selection in regression: RMSE (root-mean-squared error) or MaxAE (max absolute error).

nmodels

(R&C). **nmodels** = integer. Number of the top ranked models to output (see the folder 'models').

isconvex

(C). **isconvex** = (k,k,k,...). In classification, each data domain can be restricted to be convex or nonconvex. k takes the value of 1 or 0. 1: YES; 0: NO.

bwidth

(C). **bwidth** = float. The boundary tolerance for each domain to include the data that are outside but very close to the domain.

5.2 VarSelect_SISSO.py

If one is using the VarSelect_SISSO.py to do the VS-SISSO job, then additional control parameters can be defined in the VarSelect_SISSO.py. For details about the VS-SISSO algorithm, please refer to section 6.3.

n_init = integer # Initial size for the S .
n_RS = integer # The size of the S_a .
n_max = integer # Maximal size of the S ($S = S_a + S_b$).
nstep_max = integer # maximal iterations of the VS-SISSO
nstep_converge = integer # stop if the best model unchanged for some num. of steps.
restart = 0 or 1 # 0: start from scratch, 1: continue the job
runSISSO = '*command-to-run-SISSO*' # E.g. '*mpirun -np 64 SISSO > log*'.

6. Algorithms

6.1 Regression

As described in [Phys. Rev. Mater. 2, 083802 (2018)], the process of SISSO mainly comprises of two steps. First, feature construction. With the input variables and

mathematical operators, SISSO generates all the possible ‘meaningful’ expressions within prescribed complexity. Here, both the variables and expressions are all termed features, though with different feature complexity. The feature complexity is defined as the number of operators in a feature. Thus, all the input variables have complexity 0, and a feature such as $x + y/z$ has the feature complexity 2. All these features form a space, termed feature space.

Given the set of operators \hat{H} (e.g. $\hat{H} = \{+, -, \times, \div, \sqrt{\quad}, \exp(\quad), \log(\quad), ^2, \dots\}$) and the initial feature space Φ_0 which contains all the input variables, the feature space of tier T is defined as

$$\Phi_T \equiv \Phi_{T-1} \bigcup \hat{H}[\varphi_1, \varphi_2], \quad \forall \varphi_1, \varphi_2 \in \Phi_{T-1}, \quad (6.1)$$

where φ_1, φ_2 are features in Φ_{T-1} (for unary operators only φ_1 is considered). The average “+” and difference “−” operators apply to two features with the same unit. The size of Φ_T roughly increases with $(\#\Phi_0)^{2^T} \times (\#\hat{H}_2)^{2^{T-1}}$, where $\#\Phi_0$ and $\#\hat{H}_2$ are the numbers of elements and binary operators in Φ_0 and \hat{H}_2 , respectively. Thus, the size of feature space Φ expands rapidly with the T and the number of input features in Φ_0 .

Second, descriptor identification. The feature space $\Phi = \{\varphi_i\}$ is assumed to be a basis set to expand the target property $P = \sum_i^I \beta_i \varphi_i$, and the best K features $\{\varphi_k\}$ that $P \approx \sum_k^K \beta_k \varphi_k$, $K \ll I$, are identified via sparse regression. The $\{\varphi_k\}$ is called a K -dimension descriptor. The dimension and feature complexity are the two hyperparameters defining the overall complexity of the models. Whether accurate and low-dimensional models/descriptors can exist and be identified is critically dependent on the quality of the feature space, and hence the input features, as well as the reliability of the dataset.

More detailly, the algorithm is described in Figure 6.1. The model/descriptor dimension is a hyperparameter that users will test progressively from low to high until the leftover residual error is within quality expectation. For a given dimension n , SISSO will perform n iterations of the process of “feature space construction \rightarrow subspace selection via sure independence screening (SIS) \rightarrow descriptor identification via a sparsifying operator (SO)”. In SIS, all the features of the whole feature space are firstly standardized and ranked by $|\langle P, \varphi_i \rangle|$, and then the top $\#S_{1D}$ features are selected into the subspace S_{1D} . Note the standardization is only performed in SIS for the purpose of feature ranking, but not for feature construction and model building. Construction of the feature space is repeated in each iteration because the space is too large to be stored in the computer memory, and the selection of the SIS-subspace is performed on the fly. n iterations are required because n SIS-subspaces are needed to find the nD descriptor. Features in each SIS-subspace are selected based on the absolute of the inner product $|\langle \Delta, \varphi_i \rangle|$, where the Δ is the residual error from previous iteration (The Δ is the P for the first iteration).

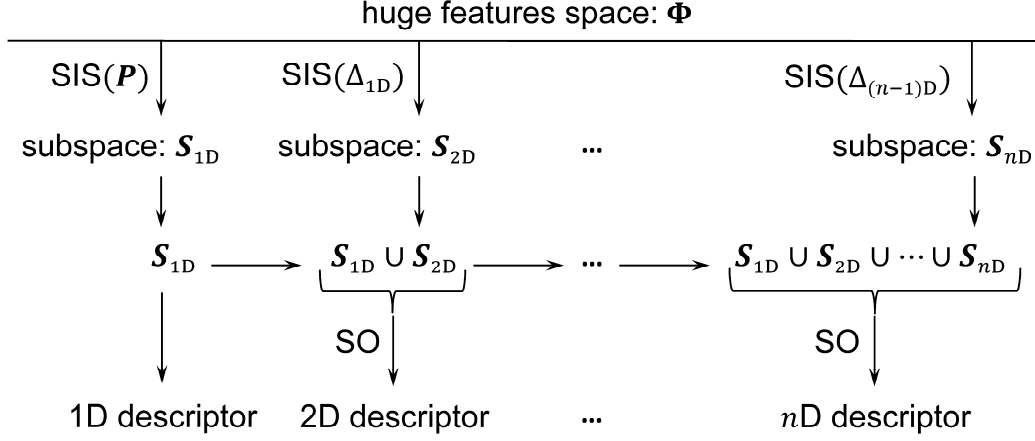


Figure 6.1. The SISSO algorithm.

In the sparse regression, ℓ_0 -norm minimization is often adopted as the SO in SISSO:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} (||\mathbf{P} - \mathbf{X}\beta||_2^2 + \lambda ||\beta||_0), \quad (6.2)$$

where \mathbf{X} is union of all the SIS-subspaces selected from the Φ , and the λ is a positive tuning hyperparameter to control the model sparsity (dimension). Though the ℓ_0 -norm minimization is a NP-hard problem, it remains computationally feasible if one focus on low-dimensional solutions, e.g. $< 5D$, with reasonable SIS-subspace size. For example, if $n = 2$, the size of each SIS-subspace can be as large as 250,000, with few hours of runtime using ~ 100 CPU cores. The total number of evaluated models by ℓ_0 is thus about $C(500000, 2) = 1.2 \times 10^{11}$. Supposing 1.2×10^{11} is the computationally allowed largest number, if $n = 3, 4, 5$, then the corresponding size of each SIS-subspace is limited to about 3000, 330, 90, respectively. For this reason, users need to **set proper SIS-subspace size for different n** . In principle, the larger the SIS-subspace, the higher probability of identifying the best models. In case $n > 10$ when each SIS-subspace can accept only 1 feature because of the limitation of computing power, SISSO becomes the orthogonal matching pursuit (OMP) method. Therefore, SISSO is a method in between the SO (SIS-subspace equal the total feature space) and the OMP (each SIS-subspace contains only 1 feature). In addition to ℓ_0 , LASSO is also available as the SO for its computational efficiency, unfortunately it often yields poor results when combining with SIS.

6.2 Classification

In classification, the SISSO framework remains the same, but the definition for the loss function and the SIS are modified. In the space of descriptors, the domain of each data group/category is approximated by convex volume, as shown in Figure 6.2. Given a property with M categories, the error $||\mathbf{P} - \mathbf{X}\beta||_2^2$ in regression is replaced by the total overlap between all the M domains. The ℓ_0 -norm minimization for classification is defined as:

$$\hat{c} = \underset{c}{\operatorname{argmin}} \left(\sum_{i=1}^{M-1} \sum_{j=i+1}^M O_{ij} + \lambda \|c\|_0 \right), \quad (6.3)$$

where O_{ij} is the number of data in the overlap region between the i and j domains, c is a sparse vector (0/1 elements) so that a feature k is selected (deselected) when $c_k = 1(0)$, and λ is a tuning hyperparameter controlling the number of nonzero elements in c .

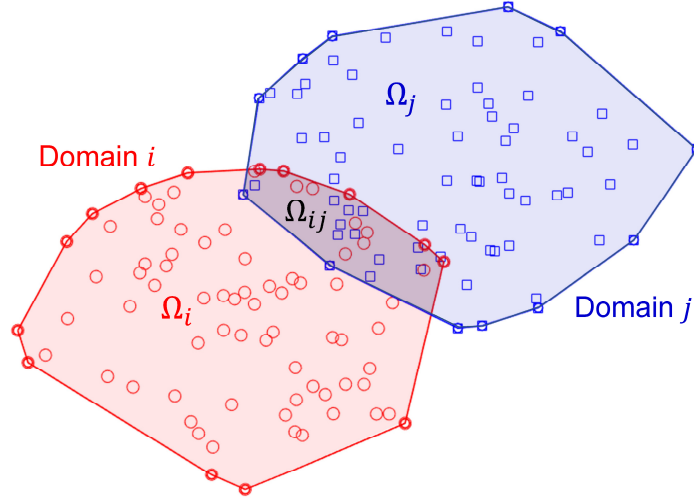


Figure 6.2. The domains for each dataset are described by convex volume. The overlap area is indicated by the Ω_{ij} .

The solutions to Equation (6.3) may not be unique. Of all the possible solutions to Equation (6.3), i.e. having the same dimension and overlap-data, the one with minimal n -dimensional overlap-volume is selected:

$$\Omega \equiv \frac{2}{M(M-1)} \sum_{i=1}^{M-1} \sum_{j=i+1}^M \frac{\Omega_{ij}}{\min(\Omega_i, \Omega_j)}, \quad (6.4)$$

where Ω_i , Ω_j , and Ω_{ij} are the n -dimensional volumes of the i , j , and overlap ij domains. Ω has values from 0 to 1. For example, with $M = 2$, if one domain is totally included in another domain, then $\Omega = 1$; if two domains are separated, $\Omega = 0$.

In SIS for classification, the correlation between the property and a feature $|\langle P, \varphi_i \rangle|$ for regression is now replaced by the $(\sum_{i=1}^{M-1} \sum_{j=i+1}^M O_{ij} + 1)^{-1}$ so that the higher the correlation, the lower the overlap between domains in the one-dimensional descriptor space. With the above new definitions for SIS and SO, SISSO can be applied to find the best descriptor that minimize the total overlap, or maximize the separation, between domains.

Note that above definition of error based on minimization of domain overlap is different from usual methods such as support vector classification and decision tree. Thus, the best classification model from SISO may not work well for other methods because of their different metrics in model selection. The difference is that, SISO not only minimize the number of misclassified data, but also minimize the misclassified domain size. With a given nD descriptor from SISO, one can straightforwardly find the corresponding separating hyperplanes and rules by using support vector classification and decision tree, respectively.

6.3 Variable selection assisted symbolic regression

As discussed in section 1.1, the size of feature space Φ_T roughly increase with $(\#\Phi_0)^{2^T}$. Thus, increasing the number of input features will lead to rapid growth of the Φ_T . In SISO, Φ_{T-1} need to be stored in memory in order to generate the Φ_T . To ensure fast computation and better scaling with the number of processors, the parallelization in SISO adopts the distributed memory system. Thus, the required memory to store the matrix of a feature space with M features and N samples in double precision, running on K CPU cores, is approximately $M \times N \times K \times 8/10^9$ GB. Further, each processor requires memory to store a SIS-subspace. As a result, to generate a Φ_3 , with available computer memory of 256 GB and hundreds of training data, the largest size of Φ_0 is often limited to be less than 30.

To remove the limit on the size of Φ_0 and avoid huge demand of memory, a new algorithm named VS-SISO that integrates SISO with iterative variable selection (VS) was developed [J. Chem. Theory Comput. 18, 4945 (2022)]. The idea of VS-SISO is shown in Figure 6.3, and the five steps are described as below.

- 1) Obtaining a subset S_a from the large pool of input features by random search, which has priority on unvisited features before all are visited once.
- 2) Let S being the union of S_a and S_b , where S_b (initially empty) is the set of variables appearing in the best model so far from symbolic regression.
- 3) Perform symbolic regression (SISO) with variables in S to update the model, which has fixed complexity in all iterations.
- 4) Update the S_b if the training accuracy of the model is improved.
- 5) Stop if the accuracy converges or the iteration hits maximum; otherwise, go to step 1.

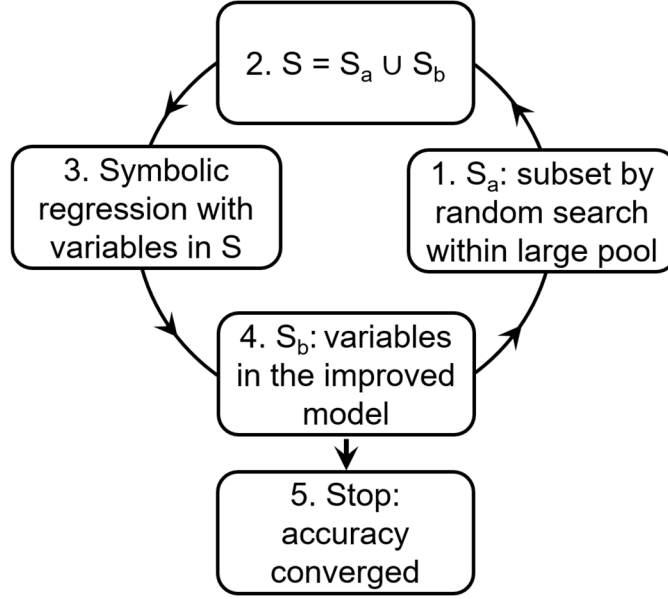


Figure 6.3. The VS-SISSO algorithm.

6.4 Multi-task learning

Multi-task learning within the framework of SISSO (MT-SISSO) was designed to identify a common n -dimensional descriptor for several related properties [J. Phys.: Mater. 2, 024002 (2019)]. Given T properties (P^t) expanded in the same feature space $\{\varphi_i\}$, $P^t = \sum_i^N \beta_i^t \varphi_i$, the outcome of MT-SISSO learning is a common sparse model, says 2D, $P^t = \beta_1^t \varphi_1 + \beta_2^t \varphi_2$. The coefficients $\{\beta_1^t, \beta_2^t\}$ are different for different t (task), but the 2D descriptor $\{\varphi_1, \varphi_2\}$ is the same for all the tasks. With the feature space $X = \{\varphi_i\}$ and sparsifying operator ℓ_0 , the problem of regression for multi-task learning can be described as:

$$\hat{\mathbf{B}} = \underset{\mathbf{B}}{\operatorname{argmin}} \sum_{t=1}^T \frac{1}{N^t} (\|\mathbf{P}^t - \mathbf{X}\boldsymbol{\beta}^t\|_2^2 + \lambda \|\mathbf{B}\|_0), \quad (6.5)$$

where \mathbf{X} are the same features for all the tasks, \mathbf{B} is the coefficients matrix in which each column is a coefficients vector $\boldsymbol{\beta}^t$, and N^t is the number of data in the t th property \mathbf{P}^t . In Equation (6.5), all the tasks are treated equally, regardless of the dataset size for each task.

The corresponding model error is defined as $\text{RMSE} = \sqrt{\sum \text{RMSE}_t^2 / T}$, where RMSE_t is the root-mean-squared error of task t . Equation (6.5) can be slightly modified so that the tasks are weighted according to their datasets size:

$$\hat{\mathbf{B}} = \underset{\mathbf{B}}{\operatorname{argmin}} \sum_{t=1}^T \frac{N^t}{\sum N^t} \cdot \frac{1}{N^t} (\|\mathbf{P}^t - \mathbf{X}\boldsymbol{\beta}^t\|_2^2 + \lambda \|\mathbf{B}\|_0)$$

$$= \operatorname{argmin}_{\mathbf{B}} \sum_{t=1}^T \frac{1}{\sum N^t} (\|\mathbf{P}^t - \mathbf{X}\mathbf{B}^t\|_2^2 + \lambda \|\mathbf{B}\|_0), \quad (6.6)$$

Corresponding model error becomes $\text{RMSE} = \sqrt{\sum \text{RMSE}_t^2 \cdot N^t / \sum N^t}$.

In the SIS step of MT-SISSO, the correlation of each feature with the target property takes the form of quadratic mean of the $|\langle P^t, \varphi_i \rangle|$ over all the tasks:

$$\theta_i = \sqrt{\sum_{t=1}^T \langle P^t, \varphi_i \rangle^2 / T}. \quad (6.7)$$

SIS ranks the features according to θ_i . Features with high θ_i are selected to form the SIS-subspaces.

Multi-task learning can also be applied to classification problems [J. Phys.: Mater. 2, 024002 (2019)]. Given T properties (tasks), the goal of multi-task learning is to find a common descriptor for T maps, in each of which the domain-overlaps are minimized. The best model is obtained by minimizing the total overlap from all the maps:

$$\hat{\mathbf{C}} = \operatorname{argmin}_{\mathbf{C}} \left(\sum_{t=1}^T \sum_{i=1}^{M^{t-1}} \sum_{j=i+1}^{M^t} O_{ij} + \lambda \|\mathbf{C}\|_0 \right). \quad (6.8)$$

The metric for SIS to evaluate the importance of a feature is defined as:

$$\theta_i = \left(\sum_{t=1}^T \sum_{i=1}^{M^{t-1}} \sum_{j=i+1}^{M^t} O_{ij} + 1 \right)^{-1}. \quad (6.9)$$

6.5 Sign-constrained multi-task learning

In multi-task learning, the sign of coefficients between different tasks can be different. For example, considering the two-task 2D model $P^t = \beta_1^t \varphi_1 + \beta_2^t \varphi_2$, the first coefficient can be $\beta_1^1 < 0$ and $\beta_1^2 > 0$ for task 1 and 2, respectively. In the scenario that when MT-SISSO was applied to inconsistent multi-source experimental data, with the P^t for different tasks being the same quantity but the corresponding datasets coming from different sources (different exp. methods), it was shown that such difference of the coefficients' signs could harm the model's interpretability and predictive performance [Wang et al., J. Am. Chem. Soc. 145,

11457 (2023)]. This difficulty was solved by adding a sign-constraint to the traditional multi-task learning problem in Equation (6.5) to ensure that the coefficients between different tasks have the same signs:

$$\hat{\mathbf{B}} = \underset{\mathbf{B}}{\operatorname{argmin}} \sum_{t=1}^T \frac{1}{N^t} \|\mathbf{P}^t - \mathbf{X}\boldsymbol{\beta}^t\|_2^2 + \lambda \|\mathbf{B}\|_0 \quad (6.10)$$

subject to: $\operatorname{sgn}(\beta_j^t \cdot \beta_j^{t'}) \geq 0 \quad \forall t, t',$

where the sgn is the sign function. This new scheme was termed sign-constrained multi-task SISSO (SCMT-SISSO). Performance of this method can be found in the work [Wang et al., J. Am. Chem. Soc. 145, 11457 (2023)].

In solving the Equation (6.10), all possible signs are tested, and the one with the lowest model error will be selected. For example, in the 2D model $P^t = \beta_1^t \varphi_1 + \beta_2^t \varphi_2$, the RMSE with $(\beta_1^t < 0, \beta_2^t < 0, \forall t)$, $(\beta_1^t < 0, \beta_2^t > 0, \forall t)$, $(\beta_1^t > 0, \beta_2^t < 0, \forall t)$, and $(\beta_1^t > 0, \beta_2^t > 0, \forall t)$ are all calculated. As a result, the time cost for the sparse regression (ℓ_0) due to the sign-constraint in SCMT-SISSO will be increased by a factor of 4.

7. Utilities

Useful tools for data processing and results analysis are provided in the folder “utilities”. Explanations on how to use these tools can be found in the file Readme.