AutoGeneS

Hananeh Aliee, Maxim Schmidt

Mar 29, 2020

CONTENTS

AutoGeneS is a tool to automatically extracts informative genes and reveals the cellular heterogeneity of bulk RNA samples. AutoGeneS requires no prior knowledge about marker genes and selects genes by simultaneously optimizing multiple criteria: minimizing the correlation and maximizing the distance between cell types. It can be applied to reference profiles from various sources like single-cell experiments or sorted cell populations.

It is compatible with scanpy. To report issues or view the code, please refer to our github page.

CHAPTER

ONE

BACKGROUND

The proposed feature selection approach solves a multi-objective optimization problem. As the name suggests, multi-objective optimization involves more than one objective function to be optimized at once. When no single solution exists that simultaneously optimizes each objective, the objective functions are said to be conflicting. In this case, the optimal solution of one objective function is different from that of the others. This gives rise to a set of trade-off optimal solutions popularly known as Pareto-optimal solutions. The list of Pareto-optimal solutions includes non-dominated solutions, which are explored so far by the search algorithm. These solutions cannot be improved for any of the objectives without degrading at least one of the other objectives. Without additional subjective preference, all Pareto-optimal solutions are considered to be equally good.

In AutoGeneS, we have n binary decision variables where n is equal to the number of genes from which the optimizer selects the markers. The value of a decision variable represents whether the corresponding gene is selected as a marker. Later, we evaluate the objective functions (correlation and distance) only for genes G whose decision variables are set to one.

AutoGeneS uses a genetic algorithm (GA) as one of the main representatives of the family of multi-objective optimization techniques. GA uses a population-based approach where candidate solutions that represent individuals of a population are iteratively modified using heuristic rules to increase their fitness (i. e., objective function values). The main steps of the generic algorithm are as follows:

- 1. Initialization step: Here the initial population of individuals is randomly generated. Each individual represents a candidate solution that, in the feature selection problem, is a set of marker genes. The solution is represented as a bit string with each bit representing a gene. If a bit is one, the corresponding gene is selected as a marker.
- 2. Evaluation and selection step: Here the individuals are evaluated for fitness (objective) values, and they are ranked from best to worst based on their fittness values. After evaluation, the best feasible individuals are then stored in an archive according to their objective values.
- 3. Termination step: Here, if the termination conditions (e. g., if the simulation has run a certain number of generations) are satisfied, then the simulation exits with the current solutions in the archive. Otherwise, a new generation is created.
- 4. If the simulation continues, the next step is creating offspring (new individuals): The general GA modifies solutions in the archive and creates offspring through random-based crossover and mutation operators. First, parents are selected among the candidates in the archive. Second, the crossover operator combines the bits of the parents to create the offspring. Third, the mutation operator makes random changes to the offspring. Offspring are then added to the population, and the GA continues with step 2.

CHAPTER

TWO

GETTING STARTED

Install AutoGeneS with:

```
pip install --user autogenes
```

In the following, we show how to use AutoGeneS with an example.

Import the libraries and read the reference data and bulk samples:

```
import anndata
import numpy as np
import pandas as pd
import autogenes as ag
import anndata

bulk_data = pd.read_csv('bulk_data.csv').transpose()
adata = sc.read(address_to_your_sc_data, cache=True).transpose()
```

Before you can use AutoGeneS, it needs to be initialized with the reference data (see the API):

```
ag.init(adata,use_highly_variable=True,celltype_key='cellType')
```

If the data is given as anndata, ag.init automatically measures the centroids of cell types by means of averaging the gene expression of their cells.

In the next step, we run the optimizer:

```
ag.optimize(ngen=5000,nfeatures=400,seed=0,mode='fixed')
```

Here, we run the optimizer for 5K generations asking for 400 genes. During this optimization process, a set of solutions is generated. Each solution is a set of 400 genes and is evaluated based on objectives that can be passed to *optimize*. In our examples, it uses the default objectives *correlation* and *distance*.

All the non-dominated solutions can be visualized using the plot function:

```
ag.plot(weights=(-1,0))
```

This plots the objective values of all non-dominated solutions. The arguments are used to select a solution, which is marked in the plot. In our case, we choose the solution using the weights *(-1,0)* on the objective values that will return the solution with minimum correlation.

To choose another solution, run:

```
ag.select(close_to=(1,75))
```

The following criterion is used to select a solution: The first number, *1* refers to the second objective, which is *distance* in our case. So, the above will choose the solution whose distance value is *closest* to 75. There are other ways of selecting a solution like specifying the index of a solution using *ag.select(index=0)*.

Now that we have chosen a solution with a set of genes, we can deconvolute bulk samples:

```
coef = ag.deconvolve(bulk_data, model='nnls')
print(coef)
```

We recommend to normalize the coeffiecients after the analysis.

It is possible to compress all of the above steps into a single line:

```
ag.pipeline(adata,bulk_data,ngen=5000,nfeatures=400,seed=0,mode='fixed',close_to=(1,
↪75),model='nnls')
```

This should produce the same result!

For more information on each step, please refer to the API. For more extensive examples, see the examples section.

CHAPTER

THREE

API

Import AutoGeneS as:

```
import autogenes as ag
```

3.1 Main functions

init(data[, celltype_key, genes_key, ...])	Preprocesses input data
optimize([ngen, mode, nfeatures, weights, ...])	Runs multi-objective optimizer
plot([objectives, weights, index, close_to])	Plots objective values of solutions
select([weights, close_to, index, copy, ...])	Selects a solution
deconvolve(bulk,key = None[, model])	Performs bulk deconvolution

3.1.1 autogenes.Interface.init

Interface.**init**(*data*, *celltype_key* = *'celltype'*, *genes_key* = *None*, *use_highly_variable* = *False*)

Preprocesses input data

If an AnnData object is passed, it is assumed that it contains single-cell data. The means are calculated using 'celltype_key'. In addition, a pre-selection of genes can be specified with 'genes_key' or 'use_highly_variable'. Then, only these genes will be considered in the optimization.

If a DataFrame or numpy array is passed, it is assumed that they already contain the means.

Parameters

- **data** (*anndata.AnnData*, *np.ndarray*, *pd.DataFrame*) – Input data
- **celltype_key** (*str*, optional (default: *celltype*)) – Name of the obs column that specifies the cell type of a cell For AnnData only
- **genes_key** (*str*, optional (default: *None*)) – Name of the var column with boolean values to pre-select genes
- **use_highly_variable** (*bool*, optional (default: *False*)) – Equivalent to genes_key='highly_variable'

Returns *None*

3.1.2 autogenes.Interface.optimize

Interface.**optimize**(*ngen = 2*, *mode = 'standard'*, *nfeatures = None*, *weights = None*, *objectives = None*, *seed = 0*, *verbose = True*, ***kwargs*)

Runs multi-objective optimizer

This method runs an evolutionary algorithm to find gene selections that optimize certain objectives. It can run for a different number of generations and in different modes. For more information on genetic algorithms and their parameters, refer to the DEAP documention.

Parameters

- **ngen** (*int*, optional (default: *2*)) – Number of generations. The higher, the longer it takes
- **mode** (*standard*, *fixed*, optional (default: *standard*)) – In standard mode, the number of genes of a selection is allowed to vary arbitrarily. In fixed mode, the number of selected genes is fixed (using *nfeatures*)
- **nfeatures** (*int*, optional (default: *int*)) – Number of genes to be selected in fixed mode
- **weights** (*(int, ...)*, optional (default: *(-1,1)*)) – Weights applied to the objectives. For the optimization, only the sign is relevant: *1* means to maximize the respective objective, *-1* to minimize it and *0* means to ignore it. The weight supplied here will be the default weight for selection. There must be as many weights as there are objectives
- **objectives** (*([str,function], ...)*, optional (default: *('correlation','distance')*)) – The objectives to maximize or minimize. Must have the same length as weights. The default objectives (correlation, distance) can be referred to using strings. For custom objectives, a function has to be passed. For further details, refer to the respective tutorial.
- **seed** (*int*, optional (default: *0*)) – Seed for random number generators
- **verbose** (*bool*, optional (default: *True*)) – If True, output a progress summary of the optimization (the current generation, size of the pareto front, min and max values of all objectives)
- **population_size** (*int*, optional (default: *100*)) – Size of every generation (mu parameter)
- **offspring_size** (*int*, optional (default: *50*)) – Number of individuals created in every generation (lambda parameter)
- **crossover_pb** (*float*, optional (default: *0.7*)) – Crossover probability
- **mutation_pb** (*float*, optional (default: *0.3*)) – Mutation probability
- **mutate_flip_pb** (*float*, optional (default: *1E-3*)) – Mutation flipping probability (fixed mode)
- **crossover_thres** (*int*, optional (default: *1000*)) – Crossover threshold (standard mode)
- **ind_standard_pb** (*float*, optional (default: *0.1*)) – Probability used to generate initial population in standard mode

Returns *None*

3.1.3 autogenes.Interface.plot

`Interface.`**`plot`**(*objectives = (0, 1)*, *weights = None*, *index = None*, *close_to = None*)

Plots objective values of solutions

Can only be run after *optimize*. Every parameter corresponds to one selection method. Only one can be chosen at a time. If you don't specify an selection method, the weights passed to *optimize* will be used.

Parameters

- **`objectives`** (*(int,int)*, optional (default: *(0,1)*)) – The objectives to be plotted. Contains indices of objectives. The first index refers to the objective that is plotted on the x-axis. For example, *(2,1)* will plot the third objective on the x-axis and the second on the y-axis.
- **`weights`** (*(int, ...)*, optional) – Weights with which to weight the objective values. For example, *(-1,2)* will minimize the first objective and maximize the the second (with higher weight).
- **`index`** (*int*, *(int,int)*, optional) – If one int is passed, return *pareto[index]* If two ints are passed, the first is an objective (*0* for the first). The second is the nth element if the solutions have been sorted by the objective in ascending order. For example, *(0,1)* will return the solution that has the second-lowest value in the first objective. *(1,-1)* will return the solution with the highest value in the second objective.
- **`close_to`** (*(int,int)*, optional) – Select the solution whose objective value is closest to a certain value. Assumes *(objective,value)*. For example, *(0,100)* will select the solution whose value for the first objective is closest to 100.

3.1.4 autogenes.Interface.select

`Interface.`**`select`**(*weights = None*, *close_to = None*, *index = None*, *copy=False*, *key_added='autogenes'*)

Selects a solution

Specify a criterion to choose a solution from the solution set. Supports adding the solution to the annotation of an adata object. Can only be run after *optimize*

Parameters

- **`weights`** (*(int, ...)*, optional) – Weights with which to weight the objective values. For example, *(-1,2)* will minimize the first objective and maximize the the second (with more weight).
- **`index`** (*int*, *(int,int)*, optional) – If one int is passed, return *pareto[index]* If two ints are passed, the first is an objective (*0* for the first). The second is the nth element if the solutions have been sorted by the objective in ascending order. For example, *(0,1)* will return the solution that has the second-lowest value in the first objective. *(1,-1)* will return the solution with the highest value in the second objective.
- **`close_to`** (*(int,int)*, optional) – Select the solution whose objective value is close to a certain value. Assumes *(objective,value)*. For example, *(0,100)* will select the solution whose value for the first objective is closest to 100.
- **`copy`** (*bool*, optional (default: *False*)) – If true, a new adata object will be created with the selected solution in the var column specified by *key_added*
- **`key_added`** (*str*, optional (default: *autogenes*)) – The name of the var column to which to add the chosen gene selection

3.1.5 autogenes.Interface.deconvolve

`Interface.`**`deconvolve`**(*bulk*, *key = None*, *model='nusvr'*)

Performs bulk deconvolution

Deconvolves bulk data using a gene selection. The selection can be specified through a key or the current selection is used.

If the optimizer has been run, but nothing has been selected yet, an automatic selection occurs (equivalent to `ag.select()`)

Parameters

- **`bulk`** (*np.ndarray*, *pd.Series*, *pd.DataFrame*, *AnnData*) – If multi-dimensional, then each row corresponds to a sample. If it has gene annotations (e.g. var_names for AnnData or df.columns for DataFrame), the method will respond intelligently (reorder if necessary, use only those genes from the selection that are available in the bulk data)
- **`key`** (*str*, optional (default: *None*)) – Name of the var column that specifies a gene selection. If None, then the current selection is used (or is automatically chosen)
- **`model`** (*nusvr*, *nnls*, *linear*, optional (default: *nusvr*)) – Choose a regression model. Available options: NuSVR, non-negative least squares and linear model.

Returns An array of the form *[[float, ...],...]* containing the model coefficients for each target (bulk sample)

3.2 Auxiliary functions

pipeline(data,bulk, **kwargs)	Runs the optimizer, selection and deconvolution using one method
resume()	Resumes an optimization process that has been interrupted
save(filename)	Saves current state to a file
load(filename)	Loads a state from a file

3.2.1 autogenes.Interface.pipeline

`Interface.`**`pipeline`**(*data*, *bulk*, ***kwargs*)

Runs the optimizer, selection and deconvolution using one method

3.2.2 autogenes.Interface.resume

Interface.**resume**()
Resumes an optimization process that has been interrupted

3.2.3 autogenes.Interface.save

Interface.**save**(*filename*)
Saves current state to a file

Parameters **filename** (*str*) – Name of the file

3.2.4 autogenes.Interface.load

Interface.**load**(*filename*)
Loads a state from a file

Parameters **filename** (*str*) – Name of the file

3.3 Accessors

adata()	Returns AnnData object
pareto()	Returns the entire solution set
fitness_matrix()	Returns fitness matrix
selection()	Returns the current selection

3.3.1 autogenes.Interface.adata

Interface.**adata**()
Returns AnnData object

Returns The AnnData object that the optimizer operates on (if no AnnData was passed to *ag.init*, *None*)

3.3.2 autogenes.Interface.pareto

Interface.**pareto**()
Returns the entire solution set

Returns The solution set in the form *[[bool],. . .]*. Every member corresponds to a gene selection

3.3.3 autogenes.Interface.fitness_matrix

Interface.**fitness_matrix**()
Returns fitness matrix

Returns A *pd.DataFrame* that contains the objective values of all solutions. The nth row corresponds to the nth solution (`ag.pareto()[n]`)

3.3.4 autogenes.Interface.selection

Interface.**selection**()
Returns the current selection

Returns *The current selection as a boolean array*

CHAPTER

FOUR

REFERENCES

[Aliee, Hananeh and Theis, Fabian, AutoGeneS: Automatic gene selection using multi-objective optimization for RNA-seq deconvolution](https://www.biorxiv.org/content/early/2020/02/23/2020.02.21.940650)

INDEX