EvolutionaryModelDiscovery Documentation

Release 0.0.4

Chathika Gunaratne

Jun 06, 2022

Contents

Automated agent rule generation and importance evaluation for agent-based models with Genetic Programming and Random Forests.

EvolutionaryModelDiscovery is a framework through which Factors affecting human decision making maybe be assessed on their importance towards the production of a social outcome.

The first phase of EvolutionaryModelDiscovery is evolving combinations of Factors to generate mechanisms for a behavior rule of interest in an ABM. For this, EvolutionaryModelDiscovery performs genetic programming on the supplied ABM, automatically generating and testing versions of the ABM with hypothetical combinations of these Factors.

Articles:

- Gunaratne, C., & Garibay, I. (2017, July). Alternate social theory discovery using genetic programming: towards better understanding the artificial anasazi. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 115-122). ACM.
- Gunaratne, C., & Garibay, I. (2018). Evolutionary Model Discovery of Factors for Farm Selection by the Artificial Anasazi. arXiv preprint arXiv:1802.00435.‘_

CHAPTER 1

Contents

1.1 Getting Started

EvolutionaryModelDiscovery (EMD for short) follows the following workflow:

1. Indentify and Tag EMD entry point (decision rule to be evolved).
2. Identify and tag your hypothesized factors
3. Intialize EMD run
4. Evolve model
5. Run and visualize Factor importance statistics (WIP)

1.1.1 1. Tag the EMD Entry Point

First the Behavior Rule of interest, to be evolved and evaluated must be tagged from within the NetLogo model. Typically this is a line of code in the NetLogo model, of which the right hand side is udetermined.

The line of code can be tagged using EMD annotations. The line of code must be sperated with a new line into its determined and undetermined components. A NetLogo comment containing EMD *Annotations* must then be instered between these components. In particular, the `@EvolveNextLine` annotation must be used to indicate that the undetermined portion of the NetLogo instruction is to be evolved and evaluated.

The example below illustrates this process. Say, for instance, how the agent variable `utility`, at line 120 of the following program, was set was undetermined.

```
117 ask turtles [
118    print (word "About to set agent " who "'s utility")
119    ; Code to set agent utility. Assuming this is a random float between 0 and 1
120    set utility random-float 1
121 ]
```

Line 120 would then have to be seperated into its determined and undetermined components with a new line. Between these two lines of code (effectively a single NetLogo instruction) we insert our EMD annotations via a NetLogo comment.

```
117 ask turtles [
118    print (word "About to set agent " who "'s utility")
119    ; Code to set agent utility. Assuming this is a random float between 0 and 1
120    set utility
121        ; @EMD @EvolveNextLine @return-type=utility-value
122        random-float 1
123 ]
```

The inserted comment (line 121) containts three EMD annotations:

- `@EMD`, which indicates that this is a series of EvolutionaryModelDiscovery annotations,
- `@EvolveNextLine`, which indicatese that the next line of code is to be evolved, and
- `@return-type=*<return-type>*`, which specifies the expected final (custom) return type of the code to be evolved in the next line.

1.1.2 2. Tag Your Factors

EMD *Factors* are implemented as NetLogo procedures. EvolutionaryModelDiscovery recognizes Factors through the `@Factor` annotation (See :ref‘Annotations‘ for full description).

By default, EvolutionaryModelDiscovery assumes that your Factors are defined on the model file itself. Instead, you may specify them on a seperated `.nls` file/s of your choice. In this case you will have to specify the `@factors-file=*<factors-file-path>*` annotation, providing the path to the `.nls` file containing the Factor specification.

At least one Factor **must** be tagged with the return type specified at the EMD entry point.

For the above example, we could, for instance define some Factors as shown below:

```
; @EMD @Factor @return-type=utility-value @parameter-type=neighbor-set
to-report utilityAsNeighborCount [neighbors]
   ; return the number of neighbors as the utility of the agent
   report count neighbors
end

; @EMD @Factor @return-type=utility-value @parameter-type=neighbor-set
to-report utilityAsNeighborEnergy [neighbors]
   ; Say all agents had a variable energy and the utility might have been the mean␣
↪energy of its neighbors
   report mean [energy] of neighbors
end

; @EMD @Factor @return-type=neighbor-set
to-report nearestNeighbors
   report turtles-on neighbors
end

; @EMD @Factor @return-type=neighbor-set
to-report linkedNeighbors
   report link-neighbors
end
```

By looking at the above example Factors, the genetic program can produce syntax trees of depth 2.

1.1.3 3. Initialize EvolutionaryModelDiscovery

Once the NetLogo model has been tagged with *Annotations* marking the EMD entry point and Factors, EvolutionaryModelDiscovery can now be initialized from Python code.

In order to initialize an EvolutionaryModelDiscovery run, the following information regarding the model and simulations need to be provided:

- The model path: The location of the `.nlogo` file containing the NetLogo to be evolved and evaluated.
- NetLogo setup commands: These commands are run during the setup of the model
- NetLogo measurement reporters: These are used to collect statistics from each simulation run and are returned to the objective function to be used by the user.
- Number of ticks to run the simulations for.

With this information, an EvolutionaryModelDiscovery run can be initialized as follows:

```
# Import EvolutionaryModelDiscovery
from EvolutionaryModelDiscovery import EvolutionaryModelDiscovery
# Provide the model path
modelPath = "SimpleSchellingTwoSubgroups_HatnaAdaption.nlogo"
# List the setup commands. In this case we just execute the model's setup procedure
setup = ['setup']
# Provide measurement reporters to evaluate the simulations. In this simple example,␣
↪we're measuring ticks and number of agents
measurements = ["ticks", "count turtles"]
# Specify how many ticks you want to run each simulation for.
ticks = 100
# Use the above information to initialize EvolutionaryModelDiscovery
emd = EvolutionaryModelDiscovery(modelPath,setup, measurements, ticks)
```

An EvolutionaryModelDiscovery object has now been created with a DEAP primitive set corresponding to the `@Factor` annotations you provided to the model.

1.1.4 4. Evolve Your Model

Once steps 1 to 3 are completed, the model is ready to be evolved. But first we need to define the objective function.

EvolutionaryModelDiscovery requires a callback function to be defined as the objective function. This function should be defined with a single parameter. When EvolutionaryModelDiscovery is run, this parameter will receive a Pandas dataframe with the simulation results. The number columns of this dataframe will be equal to the number of measurement commands EvolutionaryModelDiscovery was initialized with, and will be ordered in the order the commands were specified. The objective function should return a `float` or `int` that signifies the measured fitness of the simulation run.

For instance, for the above example:

```
# Objective function for the simulation runs
# The function has a single parameter results, to which the Pandas dataframe with the␣
↪results for the measurement commands for each simulation tick are reported in order.
def averageNumberOfAgents(results):
    # Return the mean number of agents per tick of the simulation run.
    return results.mean()[0]
# Set the objective function
emd.setObjectiveFunction(cindexObjective)
```

Additionally, we can set hyperparameters of the genetic program. For example:

```
emd.setMutationRate(0.1)
emd.setCrossoverRate(0.8)
emd.setGenerations(1)
```

The model can then be evolved using the `evolve()` command. NOTE: due to how parallelization works in Python, this function **MUST** be included in the `if __name__ == '__main__':` directive to ensure that multiple EvolutionaryModelDiscovery runs are not triggered upon parallelization.

```
if __name__ == '__main__':
    emd.evolve()
```

Parallelization

Parallelization has been paused on the latest version due to compatibility with Windows and will be reintroduced soon!

1.2 Factors

EvolutionaryModelDiscovery works by combining and re-combining Factors (implmeneted as NetLogo procedures), automatically generating NetLogo models for each combinations, and evaluating them on a modeler-defined objective function.

Factors are strongly types, i.e. the return type and parameter types must be specified. Only Factors that have a return type equal to a parameter type of another Factor may connect to the second Factor as a child Factor.

IMPORTANT: Please, ensure that there is at least one Factor with a matching return type for each parameter type in the entire set of annotated Factors. If there are Factors with parameter types for which there are no Factors with matching return types then an error will be thrown upon initialization.

1.2.1 Tagging Factors

In NetLogo, an EvolutionaryModelDiscovery factor is essentially a procedure. You can define EvolutionaryModelDiscovery ready factors by annotations in comments above your reporter.

For a full list of annotations used in EvolutionaryModelDiscovery please refer *Annotations*.

The typical Factor specification on your model should look as follows

```
;@EMD @factor @return-type=*<return-type>* @parameter-type=*<param1-type>* ...␣
↪@parameter-type=*<paramN-type>*
to-report exampleFunction [param1 ... paramN]
...
report *something*
end
```

Automatically Defined Functions

To specify an ADF, the @ADF annotation can be used. This is especially useful if a single agent behavior must be represented as the result of one or more encapsulating functions, which must themselves be evolved and evaluated through the genetic program.

```
;@EMD @factor @ADF=*<ADF-name>* @return-type=*<return-type>* @parameter-type=*<param1-
↪type>* ... @parameter-type=*<paramN-type>*
to-report exampleFunction [param1 ... paramN]
...
report *something*
end
```

In the following example, the `exampleFunction` is annotated as part of an ADF *emotions*:

```
;@EMD @factor @ADF=emotionalProcess @return-type=polarity-score @parameter-
↪type=neighborhood
to-report exampleFunction [neighbors]
   ...
end
```

1.3 Reference

1.3.1 Annotations

EvolutionaryModelDiscovery uses comment annotations when identifying parts of the NetLogo model to be evolved.

Below is a complete list of annotations used in EvolutionaryModelDiscovery:

Annotation	Example	Meaning
@EMD	@factors-file="util/Functions.nls"	Indicates the start of an EMD specification
@EvolveNextLine	@EvolveNextLine	Indicates that the next line as the entry point for evolution of the behavior rule
@factors-file	@factors-file="util/FactorsFile.nls"	
@return-type	@return-type=patchTypeA	Factor below returns something of the specified
@parameter-type	@parameter-type=emotion	Factor below takes a parameter of this type
@ADF	@ADF=emotionalProcess	ADF to which this factor belongs

1.4 License

1.5 Contact

Questions? Please contact Chathika Gunaratne <chathikagunaratne@gmail.com>

CHAPTER 2

Requirements

EvolutionaryModelDiscovery has the following requirements:

- Python 2.7 or 3.6
- NetLogo 6 or higher
- JDK 1.8 (Make sure to set the path to the jdk)

EvolutionaryModelDiscovery has several other Python package dependencies that are resolved upon installation.

CHAPTER 3

Installation

You can install EvolutionaryModelDiscovery for NetLogo with `pip`:

```
pip install evolutionarymodeldiscovery
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search