# Contents

1 Table of Contents  
1.1 Setup ................................................................. 3  
1.2 Basic Usage .......................................................... 4  
1.3 Fitting ................................................................. 4  
1.4 Estimating uncertainty ............................................... 7  
1.5 Simulating ............................................................ 9  
1.6 Plotting ............................................................... 11  
1.7 Read/Write ........................................................... 12  

2 Indices and tables .................................................... 15
A Python API for modeling statistical, high-order epistasis in large genotype-phenotype maps. Decompose genotype-phenotype maps into high-order epistatic interactions. Find nonlinearity in the genotype-phenotype map. Calculate the contributions of different epistatic orders. Estimate the importance of high-order interactions on evolution.

Currently, this package works only as an API. There is not a command-line interface, and it includes few ways to read/write the data to disk out-of-the-box. We plan to improve this moving forward. Instead, we encourage you use this package inside Jupyter notebooks.
Setup

Install

The epistasis package is not yet on Pypi. Our first official release will happen as soon as our paper is out of preprint. Until then, all our software is available on Github. You can clone from there and pip install a development version.

```
    git clone https://github.com/harmslab/epistasis
    cd epistasis
    pip install -e .
```

Dependencies

The following dependencies are required for the epistasis package. **Note:** The `gpmap` package is not yet on Pypi either. It will need to be downloaded and installed following the same procedure as above.

- **gpmap:** Module for constructing powerful genotype-phenotype map python data-structures.
- **Scikit-learn:** Simple to use machine-learning algorithms
- **Numpy:** Python’s array manipulation packaged
- **Scipy:** Efficient scientific array manipulations and fitting.

There are also some additional dependencies for extra features included in the package.

- **matplotlib:** Python plotting API.
- **ipython:** interactive python kernel.
- **jupyter notebook:** interactive notebook application for running python kernels interactively.
- **ipywidgets:** interactive widgets in python.
Basic Usage

Fitting

All models in the epistasis package inherit scikit-learn classes and follow a scikit-learn interface. If you are already familiar with the

Quick guide to epistasis models

All epistasis models inherit a Scikit-learn regressor class and decorate its methods with utilities for quantifying epistasis specifically. (All the models prepend Epistasis to the name of the scikit-learn object they inherit.) The easiest way to calculate linear, epistasis in a genotype-phenotype map is to initialize a model around a GenotypePhenotypeMap object (from the gpmap python package), and call scikit-learn’s fit method.

```python
from gpmap import GenotypePhenotypeMap
from epistasis.model import EpistasisLinearRegression

# Create a genotype-phenotype map
wildtype = "00"
genotypes = ["00", "01", "10", "11"]
phenotypes = [0, 1, 1, 4]
gpm = GenotypePhenotypeMap("00", mutations)

# Initialize an epistasis model for the genotype-phenotype map
model = EpistasisLinearRegression.from_gpm(gpm)
# Fit the epistasis model.
model.fit()
```

The GenotypePhenotypeMap becomes an attribute of the model.

When fit is called on a model, the epistasis attribute is also exposed. This attribute is an EpistasisMap object, which handles internal mapping for the epistatic coefficients and includes a set of methods that make analyzing the epistatic coefficients easy.

To get a quick look at the epistatic coefficients:

```python
>>> model.epistasis.map("keys", "values")
{
    "0": 0,
    "1": 1,
    "2": 1,
    "1,2": 2
}
```

This object includes properties such as: keys, values, and labels. It also has a get_orders method, which returns submap of epistatic coefficients with only the orders passed to it.

Fit a linear, high-order epistasis model

EpistasisLinearRegression is the base class for fitting epistasis in linear genotype-phenotype maps. It uses an ordinary least squares regression approach to estimate epistatic coefficients from a list of genotypes-phenotypes pairs. It inherits Scikit-learn’s LinearRegression class and follows the same API. (All attributes and methods are the same.) You can reference their Docs for more information about the regression aspect of these models.
The EpistasisLinearRegression class extends scikit-learn’s models to fit epistatic coefficients in genotype-phenotype maps specifically. This means, it creates its own $X$ matrix argument if you don’t explicitly pass an $X$ argument into the fit method. Further, it appends a GenotypePhenotypeMap (see gpmap package) and EpistasisMap objects to the model, making analyzing the data easier and more intuitive.

Example

```python
from epistasis.models import EpistasisLinearRegression

# Define a genotype-phenotype map
wildtype = "00"
genotypes = ["00", "01", "10", "00"]
phenotypes = [0.0, 0.5, 0.5, 1.0]
order = 2

# Initialize a model
model = EpistasisLinearRegression(wildtype, genotypes, phenotypes)

# Fit the model
model.fit()

# Print the epistatic coefs
print(model.epistasis.values)
```

Fit a nonlinear epistasis model

Often, the genotype-phenotype map is nonlinear. That is to say, the genotypes and phenotypes change on different scales. Genotypes, for example, differ by discrete, linear changes in sequences (known as mutations). How these changes translate to phenotype may be less obvious. Sometimes, the effects of mutations simply add together. Sometimes, the effects multiply. Other times, they change in some other nonlinear way that is not known a priori. To estimate epistatic coefficients, the genotype-phenotype map must be on a linear scale.

EpistasisNonlinearRegression class enables you to estimate the scale of any arbitrary genotype-phenotype map. Simply define the nonlinear relationship you’d expect, or use some reasonable function that evaluates the shape (i.e. a Box-Cox transform). The EpistasisNonlinearRegression will regress this relationship using a nonlinear least squares regression (using scipy’s curve_fit function), effectively minimizing epistasis that might arise from the nonlinear relationship. It can, then, compute the linearized phenotypes.

Example

```python
from epistasis.models import NonlinearEpistasisRegression

# Define the nonlinear relationship and it's inverse.
def boxcox(x, lmbda):
    return (x**lmbda - 1) / lmbda
def reverse_boxcox(y, lmbda):
    return (lmbda*y + 1) ** (1/lmbda)

# Initialize the model
model = NonlinearEpistasisRegression.from_json("data.json",
                                          order=1,
                                          function=boxcox,
                                          )
```
The `epistasis` package also ships with widgets (via `ipywidgets`) that aid in guessing initial values for the nonlinear fit. This is incredibly useful if you are finding that the nonlinear model isn’t converging, or is converging to a local minimum in the parameter space.

```python
model.fit(lmbda=3, use_widgets=True)
```

## Fit a multiplicative, high-order epistasis model

Multiplicative epistasis (the effects of mutations multiply together) is a common nonlinear, phenotypic scale. The following example shows how to estimate epistasis from a multiplicative scale, using a simple trick of exponentials and logarithms.

\[
to
\]

\[
p = \beta_1 \beta_2 \beta_{1,2}
\]

\[
p = e^{ln(\beta_1 \beta_2 \beta_{1,2})}
\]

\[
p = e^{(ln(\beta_1 + ln(\beta_2 + ln(\beta_{1,2})))}
\]

\[
p = e^{(\alpha_1 + \alpha_2 + \alpha_{1,2})}
\]

where \( e^\alpha = \beta \)
Example

```python
import numpy as np
from epistasis.models import NonlinearEpistasisRegression

# Define the nonlinear relationship and its inverse.
def exp(x):
    return np.exp(x)
def log(y):
    return np.log(y)

# Initialize the model
model = NonlinearEpistasisRegression.from_json("data.json",
    order=1,
    function=exp,
    reverse=log
)

# Fit
model.fit()

# print multiplicative coefficients
alphas = model.epistasis.values
betas = np.exp(alphas)
```

Estimating uncertainty

The epistasis package includes a sampling module for estimating uncertainty in all parameters in a (Non)linear epistasis models. All Sampler objects create a database folder with the epistasis model stored inside a pickle file and an HDF5 file containing samples used to estimate uncertainty.

The module include two types of samplers:

1. BayesianSampler
2. BootstrapSampler

Both samplers have the same methods and attributes. They differ in their philosophy of sampling a model. See the conversation between Frequentism and Bayesianism in this blog.

Basic Example

```python
# Imports
import matplotlib.pyplot as plt
import numpy as np
import corner

from epistasis.simulate import LinearSimulation
from epistasis.models import EpistasisLinearRegression
from epistasis.sampling.bayesian import BayesianSampler

# Create a simulated genotype-phenotype map with epistasis.
sim = LinearSimulation.from_length(4, model_type="local")
sim.set_coefs_order(4)
```
sim.set_coefs_random((-1,1))
sim.set_stdeviations([0.01])

# Initialize an epistasis model and fit a ML model.
model = EpistasisLinearRegression.from_gpm(sim, order=4, model_type="local")
model.fit()

# Initialize a sampler.
fitter = BayesianSampler(model)
fitter.add_samples(500)

# Plot the Posterior
fig = corner.corner(bayes.coefs.value, truths=sim.epistasis.values)
Defining a prior

The default prior for a BayesianSampler is a flat prior (BayesianSampler.lnprior() returns a log-prior equal to 0). To set your own prior, define your own function that called lnprior that returns a log prior for a set of coefs and reset the BayesianSampler static method:

```python
def lnprior(coefs):
    # Set bound on the first coefficient.
    if coefs[0] < 0:
        return -np.inf
    return 0

# Apply to fitter from above
fitter.lnprior = lnprior
```

API

BayesianSampler

BootstrapSampler

Simulating

Epistasis can take many forms. It can be nonlinear and/or high-order (i.e. higher than three-way interactions). The epistasis package provides flexibles classes build simulated genotype-phenotype maps that exhibit such features. A few out-of-box classes examples are shown below.

Simulate a linear epistatic genotype-phenotype map

The following examples show a variety ways to simulate a genotype-phenotype map with linear, high-order epistatic interactions. The simulation interface provides methods to easily dictate the construction of a simulated genotype-phenotype map.

```python
from epistasis.simulate import AdditiveSimulation

# Define the wildtype sequence and possible mutations at each site.
wildtype = "0000"
mutations = {
    0: ["0", "1"],  # mutation alphabet for site 0
    1: ["0", "1"],
    2: ["0", "1"],
    3: ["0", "1"]
}

# Initialize a simulation
gpm = AdditiveSimulation(wildtype, mutations)

gpm.set_coefs_order(4)

gpm.set_coefs_random(coef_range)
```

1.5. Simulating
Alternatively, you can quickly simulate a binary genotype-phenotype map if you’re fine with a simple, binary alphabet at each site.

```python
# define the length of genotypes and the order of epistasis
length = 4
gpm = AdditiveSimulation.from_length(length)

# Generate random epistatic coefs
# Define the length of genotypes and the order of epistasis
gpm.set_coefs_order(4)
gpm.set_coefs_random(coef_range)
```

For all simulated genotype-phenotype maps, one can initialize a genotype-phenotype map from an existing dataset. Scroll through class methods that start with `from_` to see all options for initializing simulated genotype-phenotype maps.

### Simulate a nonlinear genotype-phenotype map

Often, the genotype-phenotype map is nonlinear. That is to say, the genotypes and phenotypes change on different scales. Genotypes, for example, differ by discrete, linear changes in sequences (known as mutations). How these changes translate to phenotype may be less obvious. Sometimes, the effects of mutations simply add together. Sometimes, the effects multiply. The `epistasis` package has `NonlinearSimulation` class that allows you to construct these more complicated genotype-phenotype maps. Simply define a function which transforms a linear genotype-phenotype map onto a nonlinear scale. Note, the function must have `x` as the first argument. This argument represents the linearized phenotypes to be transformed.

```python
from epistasis.simulate import NonlinearSimulation
def saturating_scale(x, K):
    return ((K+1)*x)/(K+x)

# Define the initial value for the parameter, K
p0 = [2]
gpm = NonlinearSimulation.from_length(4, function=saturating_scale, p0=p0)
gpm.set_coefs_order(4)
gpm.set_coefs_random((0,1))
```

### Simulate a multiplicative epistatic genotype-phenotype map

Multiplicative epistasis is a common nonlinear, phenotypic scale. You can simulate a multiplicative genotype-phenotype map using the `NonlinearSimulation` class. Remember, the `epistasis` package always constructs/decomposes epistasis as a sum of epistatic coefficients. To construct a multiplicative map, then, simply use the exponential logarithm of the epistatic coefficients:

\[ p = e^{\ln(\beta_1) + \ln(\beta_2) + \ln(\beta_{1,2})} \]
\[
\frac{\beta_1 \beta_2 \beta_{1,2} p}{e^{\ln(\beta_1) + \ln(\beta_2) + \ln(\beta_{1,2})}}
\]

Using the `epistasis` package, this looks like the following example. First, define the exponential function as the nonlinear scale passed into the Simulation class.

```python
import numpy as np
from epistasis.simulation import NonlinearSimulation

def multiplicative(x):
    return np.exp(x)

gpm = NonlinearSimulation.from_length(4, function=multiplicative)
```

Then, define the epistatic coefficients, take their log, and pass them into the simulation object.

```python
# Set the order of epistasis
gpm.set_coefs_order(4)

# generate random coefs
coefs = np.random.uniform(0,3, size=len(gpm.epistasis.labels))

# Take the log of the coefs
log_coefs = np.log(coefs)

# Pass coefs into the simulation class.
gpm.set_coefs_values(log_coefs)
```

### Plotting

The `epistasis` package comes with a few functions to plot epistasis data.

#### Coefficients

The plotting module comes with a default function for plotting epistatic coefficients. It plots the value of the coefficient as bar graphs, the label as a box plot (see example below), and significance as stars using a t-test.

```python
from epistasis.models import EpistasisLinearRegression
from epistasis.plots import coefs

# Fit with a model.
model = EpistasisLinearRegression.from_json("data.json", order=5)
model.fit()

# plot the epistasis coefs
```
labels = model.interactions.labels
values = model.interactions.values
fig, ax = coefs(labels, values)

Figure

API

Read/Write

Accepted formats

Currently, the epistasis package only takes JSON formatted data as input to the epistasis model. We expect to get include more formats (i.e. csv/excel) moving forward, just haven’t gotten around to it.

The only keys recognized by the json reader are:

1. genotypes
2. phenotypes
3. stdeviations
4. mutations
5. n_replicates
6. log_transform

All other keys are ignored in the epistasis models. You can keep other metadata stored in the JSON, but it won’t be appended to the epistasis model object.

```json
{
    "genotypes" : [
        '000',
        '001',
        '010',
        '011',
        '100',
```


```
'101',
'110',
'111'
],

"phenotypes" : [
  0.62344582,
  0.87943151,
  -0.11075798,
  -0.59754471,
  1.4314798,
  1.12551439,
  1.04859722,
  -0.27145593
],

"stdeviations" : [
  0.01,
  0.01,
  0.01,
  0.01,
  0.01,
  0.01,
  0.01,
  0.01,
],

"mutations" : {
  0 : ["0", "1"],
  1 : ["0", "1"],
  2 : ["0", "1"],
}

"n_replicates" : 12,
"log_transform" : false,
"title" : "my data",
"description" : "a really hard experiment"
```

1.7. Read/Write

13
CHAPTER 2

Indices and tables

- genindex
- modindex
- search