# mihifepe Documentation

***Release 0.2.2***

**Akshay Sood**

**Mar 30, 2020**

# Contents:

CHAPTER 1

# mihifepe

## 1.1 Overview

`mihifepe`, or **M**odel **I**nterpretability via **Hi**erarchical **Fe**ature **Pe**rturbation, is a library implementing a model-agnostic method that, given a learned model and a hierarchy over features, (i) tests feature groups, in addition to base features, and tries to determine the level of resolution at which important features can be determined, (ii) uses hypothesis testing to rigorously assess the effect of each feature on the model's loss, (iii) employs a hierarchical approach to control the false discovery rate when testing feature groups and individual base features for importance, and (iv) uses hypothesis testing to identify important interactions among features and feature groups. `mihifepe` is based on the following paper:

Lee, Kyubin, Akshay Sood, and Mark Craven. 2019. "Understanding Learned Models by Identifying Important Features at the Right Resolution." In Proceedings of the AAAI Conference on Artificial Intelligence, 33:4155–63. https://doi.org/10.1609/aaai.v33i01.33014155.

## 1.2 Documentation

https://mihifepe.readthedocs.io

## 1.3 Installation

Recommended installation method is via virtual environments and pip. In addition, you also need graphviz installed on your system.

When making the virtual environment, specify python3 as the python executable (python3 version must be 3.5+):

```
mkvirtualenv -p python3 mihifepe_env
```

To install the latest stable release:

```
pip install mihifepe
```

Or to install the latest development version from GitHub:

```
pip install git+https://github.com/Craven-Biostat-Lab/mihifepe.git@master#egg=mihifepe
```

On Ubuntu, graphviz may be installed by:

```
sudo apt-get install graphviz
```

## 1.4 Development

https://mihifepe.readthedocs.io/en/latest/contributing.html

## 1.5 Usage

https://mihifepe.readthedocs.io/en/latest/usage.html

## 1.6 License

`mihifepe` is free, open source software, released under the MIT license. See LICENSE for details.

## 1.7 Contact

Akshay Sood

# CHAPTER 2

# Usage

`mihifepe` is invoked as follows:

```
python -m mihifepe -data_filename <data_filename.hdf5>
                   -model_generator_filename <model_generator_filename.py>
                   -hierarchy_filename <hierarchy_filename.csv>
                   -output_dir <output_dir>
```

Performance can be greatly improved by running on a distributed system. Presently `mihifepe` only supports HTCondor running on a shared filesystem, invoked as follows:

```
python -m mihifepe -condor ...
```

To see a complete list of options, run:

```
python -m mihifepe -h
```

## 2.1 Inputs

- **Test data**: <*data_filename.hdf5*> Test data in HDF5 format
- **Trained model**: <*model_generator_filename.py*> Python script that generates model object for subsequent callbacks to model.predict and model.loss
- **Hierarchy over features**: <*hierarchy_filename.csv*> CSV specifying hierarchy over features

See *Input specification* for a detailed descriptions of the input data.

## 2.2 Outputs

- <*output_dir*>*/p_values.csv*: CSV, listing for all nodes in hierarchy:

  - Accuracy of model with given node perturbed
  - p-values of paired statistical test comparing perturbed model loss to baseline (unperturbed) model loss
- *<output_dir>/hierarchical_fdr_control/tree.png*: PNG showing subtree of hierarchy corresponding to rejected nodes, subsequent to hierarchical FDR control

## 2.3 Input specification

### 2.3.1 Data types

The library deals with models that accept one or both of the following types of input:

- **Static input**: Represented as a single vector per instance of length $L$, cumulatively represented as a data matrix
- **Temporal input**: Represented as an input sequence of variable length $V$, each element of which is a vector of fixed length $W$.

Models commonly take only static input, but models such as Recurrent Neural Networks (RNNs) work with input sequences. Models comprising bigger networks with RNN sub-networks may take both kinds of inputs.

### 2.3.2 Data representation

The data must be in HDF5 format, which (among other things) allows easy and scalable storage and access of a combination of static and variable-length temporal data. The recommended method of generating HDF5 inputs is via h5py.

HDF5 data is organized into a hierarchy comprising groups (containers) and datasets (data collections). The input data for `mihifepe` must be organized as follows (see https://mihifepe.readthedocs.io/en/latest/examples.html for examples).

Groups:

```
/temporal                (Group containing temporal data)
```

Datasets:

```
/record_ids              (List of record identifiers (strings) of length M = number of␣
↪records/instances)
/targets                 (vector of target values (regression/classification outputs)␣
↪of length M)
/static                  (matrix of static data of size M x L)
/temporal/<record_id>    (One dataset per record_id) (List (of variable length V) of␣
↪vectors (of fixed length W))
```

**[TODO]: Sparse representations may be used for both temporal and static data, in which case the attribute 'sparse' must be specified.**

### 2.3.3 Trained model

The caller must create a `model` object, corresponding to the trained model, that implements the following methods:

```
model.predict(target, static data, temporal_data)
"""
Predicts the model's output (loss, prediction) for the given target and instance.

Args:
    target:         classification label or regression output (scalar value)
    static_data:    static data (vector)
    temporal_data:  temporal data (matrix, where number of rows are variable across␣
↪instances)

Returns:
    loss:           model's output loss
    prediction:     model's output prediction, only used for classifiers
"""

model.loss(prediction, target)
"""
The model's loss function applied to an output/target pair for a single input␣
↪instance.
For instance, if the loss function is RMSE, the function would return
sqrt(mean(prediction - target)**2)) = abs(prediction - target)

Args:
    prediction:     model's output prediction on a single input instance
    target:         corresponding target label for instance

Returns:
    loss:           model's output loss
"""
```

This object must be generated by a standalone Python script that is passed to `mihifepe`. This allows `mihifepe` to distribute the feature perturbations across multiple worker nodes, each with its own copy of `model`. For instance, if the script path is */a/b/c/d/gen_model.py*, then `mihifepe` will access `model` as follows:

```
sys.path.insert(0, "/a/b/c/d/") # Makes python search this folder for modules
from gen_model import model
```

The test data type must match the data type of the *predict* function (e.g. if the model requires both static and temporal input, the input test data must provide both for every instance).

### 2.3.4 Hierarchy over features

The caller must provide a hierarchy over features as a CSV file. Each node (including leaf nodes) may correspond to a single feature or a group of features. Two sets of indices must be specified for each leaf node, at least one of which must be non-empty. Indices of the same data type must be mutually exclusive across leaf nodes. The CSV must contain the following columns:

```
name:             feature name, unique across features
parent_name:      name of parent if it exists, else '' (root node)
description:      node description
static_indices:   [only required for leaf nodes] list of tab-separated indices␣
↪corresponding to the indices
                    of these features in the static data
temporal_indices: [only required for leaf nodes] list of tab-separated indices␣
↪corresponding to the indices
                    of these features in the temporal data
```

CHAPTER 3

# Examples

TODO

CHAPTER 4

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 4.1 Types of Contributions

### 4.1.1 Report Bugs

Report bugs at https://github.com/Craven-Biostat-Lab/mihifepe/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

### 4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

### 4.1.4 Write Documentation

mihifepe could always use more documentation, whether as part of the official mihifepe docs, in docstrings, or even on the web in blog posts, articles, and such.

### 4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Craven-Biostat-Lab/mihifepe/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

## 4.2 Get Started!

Ready to contribute? Here's how to set up *mihifepe* for local development.

1. Fork the *mihifepe* repo on GitHub.
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/mihifepe.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
mkvirtualenv -p python3 mihifepe
cd mihifepe/
pip install -r requirements_dev.txt -r requirements.txt
```

4. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the pylint/flake8 and the tests for all python versions:

```
tox
```

   You may use pyenv to install and test with multiple python versions.

   While testing with a specific python version, you may invoke the tests as follows:

```
pytest tests
```

   To profile the code, add the –profile-svg flag (by default, the profiling results are saved in the 'prof' directory):

```
pytest --profile-svg tests
```

   If the change affects the distributed (HTCondor) implementation, you should also run condor tests in an environment that supports condor with a shared filesystem (these tests are disabled by default):

```
pytest --basetemp=condor_test_outputs tests/condor_tests
```

When writing new tests, corresponding tests and gold (expected) files for condor may be generated automatically using the following script:

```
python -m mihifepe.gen_condor_tests
```

If regression tests fail because the new data is correct, you can use the –force-regen flag to update the gold file (see pytest-regressions):

```
pytest --force-regen
```

Note: Most regression tests perform two comparisons - the output p-values and the FDR output, so the tests must be run with the –force-regen flag twice to update both the gold files.

Condor gold files may be overwritten using –force-regen as well, or simply copied over by running:

```
python -m tests.gen_condor_tests -overwrite_golds
```

6. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6 and 3.7. Check https://travis-ci.org/Craven-Biostat-Lab/mihifepe/pull_requests and make sure that the tests pass for all supported Python versions.

## 4.4 Tips

To run a subset of tests:

```
pytest tests/test_mihifepe.py  # Only run tests from specific file
pytest -k test_simulation_interactions tests/test_mihifepe.py  # Only run specific␣
↪test from given file
```

To run debugger within pytest:

```
pytest --trace  # Drop to PDB at the start of a test
pytest --pdb  # Drop to PDB on failures
```

To run pylint:

```
pylint mihifepe tests
```

To run flake8:

```
flake8 mihifepe tests
```

## 4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in CHANGELOG.rst). Then run:

```
bumpversion patch # possible: major / minor / patch
git push
git push --tags
```

Travis will then deploy to PyPI if tests pass.

# CHAPTER 5

# Credits

This library is based on the following paper:

Lee, Kyubin, Akshay Sood, and Mark Craven. 2019. "Understanding Learned Models by Identifying Important Features at the Right Resolution." In Proceedings of the AAAI Conference on Artificial Intelligence, 33:4155–63. https://doi.org/10.1609/aaai.v33i01.33014155.

## 5.1 Development Lead

- Akshay Sood <sood.iitd@gmail.com>

## 5.2 Acknowledgements

This package was refined using Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 6

# Changelog

## 6.1 0.2.1 (2019-12-29)

- Package sub-modules
- Fix Travis auto-deployment to PyPI
- Upgrade numpy dependency

## 6.2 0.2.0 (2019-12-27)

- Regression tests - serial and distributed (condor)
- Sympy to manage simulated model
- Pairwise interaction analysis
- Corrected adjusted p-values for non-rejected nodes
- Various minor fixes and documentation updates

## 6.3 0.1.1 (2018-09-14)

- First release on PyPI

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search