humanmodels

Release 0.5.7

Alberto Tonda

May 29, 2021

CONTENTS:

This package provides human-designed, scikit-learn compatible models for classification and regression. `humanmodels` are initialized through a sympy-compatible text string, describing an equation (e.g. "y = 4*x* + *3*z**2 + p_0") or a rule for classification that must return True or False (e.g. "x > 2*y + 2"). If the string contains parameters not corresponding to problem variables, the parameters of the model are optimized on training data, using the `.fit(X,y)` method.

The objective of HumanModels is to provide a scikit-learn integrated way of comparing human-designed models to machine learning models.

CHAPTER

ONE

INSTALLING THE PACKAGE

On Linux, HumanModels can be installed through pip:

```
pip install humanmodels
```

You can also install the package by cloning or downloading this repository, `cd` into the directory and then execute:

```
python -m build
python -m pip install dist/humanmodels*whl
```

On Windows, HumanModels can be installed through the Anaconda Prompt:

```
pip install humanmodels
```

CHAPTER

TWO

EXAMPLES

2.1 HumanRegressor

`HumanRegressor` is a regressor, initialized with a sympy-compatible text string describing an equation, and a dictionary mapping the correspondance between the variables named in the equation and the features in `X`. Let's generate some data to test the algorithm:

```
import numpy as np
print("Creating data...")
X_train = np.zeros((100,3))
X_train[:,0] = np.linspace(0, 1, 100)
X_train[:,1] = np.random.rand(100)
X_train[:,2] = np.linspace(0, 1, 100)
y_train = np.array([0.5 + 1*x[0] + 1*x[2] + 2*x[0]**2 + 2*x[2]**2 for x in X_train])
```

An example of initialization:

```
from humanmodels import HumanRegressor
model_string = "y = 0.5 + a_1*x + a_2*z + a_3*x**2 + a_4*z**2"
variables_to_features = {"x": 0, "z": 2}
regressor = HumanRegressor(model_string, variables_to_features)
print(regressor)
```

Printing the model as a string will return:

```
Model not initialized, call '.fit(X, y)'
```

We can now fit the model to the data:

```
print("Fitting data...")
regressor.fit(X_train, y_train)
print(regressor)
```

The code will produce:

```
Fitting data...
Model: y = a_1*x + a_2*z + a_3*x**2 + a_4*z**2 + 0.5
Variables: ['x', 'z']
Parameters: {'a_1': 1.0000001886557832, 'a_2': 1.0000004533354703, 'a_3': 2.
↪000000577731051, 'a_4': 2.0000005553527895}
Trained model: y = 2.0*x**2 + 1.0*x + 2.0*z**2 + 1.0*z + 0.5
```

As the only variables provided in the `variables_to_features` dictionary are named `x`, and `z`, all other alphabetic symbols (`a_1`, `a_2`, `a_3`, `a_4`) are interpreted as trainable parameters. The model also shows the optimized

values of its parameters. Let's now check the performance on the training data:

```
y_pred = regressor.predict(X_train)
from sklearn.metrics import mean_squared_error
print("Mean squared error:", mean_squared_error(y_train, y_pred))
```

```
Mean squared error: 7.72490931190691e-13
```

The regressor can also be tested on unseen data, and since in this case the equation used to generate the data has the same structure as the one given to the regressor, the generalization is of course satisfying:

```
X_test = np.zeros((100,3))
X_test[:,0] = np.linspace(1, 2, 100)
X_test[:,1] = np.random.rand(100)
X_test[:,2] = np.linspace(1, 2, 100)
y_test = np.array([0.5 + 1*x[0] + 1*x[2] + 2*x[0]**2 + 2*x[2]**2 for x in X_test])
y_pred = regressor.predict(X_test)
print("Mean squared error on test:", mean_squared_error(y_test, y_pred))
```

```
Mean squared error on test: 1.2055817248044523e-11
```

2.2 HumanClassifier

`HumanClassifier` also takes in input a sympy-compatible string (or dictionary of strings), defining a logic expression that can be evaluated to return `True` or `False`. If only one string is provided during initialization, the problem is assumed to be binary classification, with `True` corresponding to Class 0 and `False` corresponding to Class 1. Let's test it on the classic `Iris` benchmark provided in `scikit-learn`, transformed into a binary classification problem.

```
from sklearn import datasets
X, y = datasets.load_iris(return_X_y=True)
for i in range(0, y.shape[0]) : if y[i] != 0 : y[i] = 1

from humanmodels import HumanClassifier
rule = "(sl < 6.0) & (sw > 2.7)"
variables_to_features = {"sl": 0, "sw": 1}
classifier = HumanClassifier(rule, variables_to_features)
print(classifier)
```

```
Model not initialized, call '.fit(X, y)'
```

Even if there are no trainable parameters, the classifier must still be trained using `.fit(X,y)`, for compatibility with the `scikit-learn` package:

```
classifier.fit(X, y)
print(classifier)
```

```
Classifier: Class 0: (sw > 2.7) & (sl < 6.0); variables: sl -> 0 sw -> 1; parameters:␣
↪None
Default class (if all other expressions are False): 1
```

And now, let's test the classifier:

```
y_pred = classifier.predict(X)
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y, y_pred)
print("Final accuracy for the classifier is %.4f" % accuracy)
```

```
Final accuracy for the classifier is 0.9067
```

For multi-class classification problems, `HumanClassifier` can accept a dictionary of logic expressions in the form `{label0 : "expression0", label1 : "expression1", ...}`. As for `HumanRegressor`, expression can also have trainable parameters, optimized when `.fit(X,y)` is called. Let's see an another example with `Iris`, this time using all three classes:

```
X, y = datasets.load_iris(return_X_y=True)
rules =     {0: "sw + p_0*sl > p_1",
        2: "pw > p_2",
        1: ""}  # this means that a sample will be associated to class 1 if both
                # the expression for class 0 and 2 return 'False'
variables_to_features = {'sl': 0, 'sw': 1, 'pw': 3}
classifier = HumanClassifier(rules, variables_to_features)

classifier.fit(X, y)
print(classifier)
y_pred = classifier.predict(X)
accuracy = accuracy_score(y, y_pred)
print("Classification accuracy: %.4f" % accuracy)
```

```
Class 0: p_0*sl + sw > p_1; variables:sl -> 0 sw -> 1; parameters:p_0=-0.
↪6491880968641275 p_1=-0.12490468490418744
Class 2: pw > p_2; variables:pw -> 3; parameters:p_2=1.7073348596674072
Default class (if all other expressions are False): 1
Classification accuracy: 0.9400
```

CHAPTER

THREE

DEPENDS ON

numpy (for fast computations)

sympy (for symbolic mathematics)

scipy (for optimization)

cma (also for optimization of non-convex functions)

scikit-learn (for quality metrics, such as accuracy and mean squared error; also, HumanClassifier and HumanRegressor have the ambition of being compatible with scikit-learn estimators)

3.1 humanmodels package

3.1.1 Submodules

3.1.2 humanmodels.HumanClassifier module

class humanmodels.HumanClassifier.**HumanClassifier**(*logic_expression: Union[str, dict]*, *map_variables_to_features: dict*, *random_state=None*)

Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

Human-style classification, using a dictionary of rules to be evaluated as logic expressions (e.g. "x*2 + 4*z > 0"), to then associate samples to a class.

Builder for the class.

Parameters

- **logic_expression** (*str or dictionary {int: string}*) – A string (or dictionary of strings).
- **map_variables_to_features** (*dict*) – Dictionary containing the mapping between variables and features indexes (in datasets).
- **target_class** (*list of int, optional*) – If several logic expressions are specified, a list of target classes must be passed as an argument, in order for HumanClassification to behave as a one-vs-all classifier. The default is None.

Returns

Return type None.

check_parameters()

Check coherence of the parameters.

Returns

Return type None.

error_function(*parameter_values*, *parameter_names*, *X*, *y*)
Error function, to be optimized. The parameters in each expression given for each class are replaced by the candidate parameter values, and the predictions of the model are then compared against class labels, obtaining a cost function value depending on the results.

fit(*X*, *y*, *optimizer: str = 'cma'*, *optimizer_options: Optional[dict] = None*, *n_jobs: int = 0*, *verbose: bool = False*)
Fits the internal model to the data, using features in X and known values in y.

Parameters

- **X**(*array, shape(n_samples, n_features)*) – Training data
- **y**(*array, shape(n_samples, 1)*) – Training values for the target feature/variable
- **optimizer**(*string, default="cma"*) –

  The optimizer that is going to be used. Acceptable values:

  - ”cma”: Covariance-Matrix-Adaptation Evolution Strategy, derivative-free optimization.
- **optimizer_options**(*dict, default=None*) – Dictionary of options that can be passed to the optimization algorithm. Shape and type depend on the choice made for ‘optimizer’.
- **n_jobs**(*int, default=0*) – Option that can be passed to the optimization algorithm, number of jobs to be executed in parallel. Default is zero, avoids the use of multiprocessing. -1 uses all available CPUs.
- **verbose**(*bool, default=False*) – If True, prints internal output to screen.

Returns

Return type Self.

parameters_to_string(*c*)

predict(*X*)
Predict the class labels for each sample in X

Parameters **X** (*array, shape(n_samples, n_features)*) – Array containing samples, for which class labels are going to be predicted.

Returns **C** – Prediction vector, with a class label for each sample.

Return type array, shape(n_samples)

to_string()

variables_to_string(*c*)

3.1.3 humanmodels.HumanRegressor module

Created on Fri Apr 23 21:06:03 2021

TODO: scikit-learn compliant estimators DO NOT check the coherence of their hyperparameters in __init__ , but wait for the user to call .fit (for compatibility with grid search algorithms).

@author: Alberto Tonda

class humanmodels.HumanRegressor.**HumanRegressor**(*equation_string: str*, *map_variables_to_features: Optional[dict] = None*, *target_variable_string: Optional[str] = None*, *random_state=None*)

Bases: sklearn.base.BaseEstimator, sklearn.base.RegressorMixin

Human-designed regressor, initialized with a sympy-compatible text string describing an equation. Also needs a dictionary mapping the correspondance between the variables named in the equation and the features in X.

Builder for the class.

Parameters

- **equation_string** (*str*) –

  String containing the equation of the model. Examples:

  1. "y = 2*x + 4"
  2. "4*x_0 + 5*x_1 + 6*x_2"

  If a left-hand side variable is NOT provided (as in example #2), the optional target_variable parameter must be specified.

- **map_features_to_variables** (*dict*) – Maps the names (or integer indexes) of the features to the variables in the internal symbolic expression representing the model.

- **target_variable_string** (*str, optional*) – String containing the name of the target variable. It's not necessary to specify target_variable if the left-hand part of the equation has been provided in equation_string. The default is None.

Returns

Return type None.

check_parameters()

Checks internal parameters for consistency, before starting data fitting.

Returns

Return type None.

error_function(*parameter_values*, *X*, *y*, *verbose=False*)

Error function to be optimized. Inside the error function, the local sympy expression will have its parameters replaced with candidate values.

Returns **mse** – Mean squared error of the model's prediction with the candidate parameters, agains true values in 'y'.

Return type float

fit(*X*, *y*, *map_variables_to_features: Optional[dict] = None*, *optimizer_options: Optional[dict] = None*, *optimizer: str = 'bfgs'*, *n_jobs: int = 0*, *verbose: bool = False*)

Fits the internal model to the data, using features in X and known values in y.

Parameters

- **X** (*array, shape(n_samples, n_features)*) – Training data
- **y** (*array, shape(n_samples, 1)*) – Training values for the target feature/variable
- **map_features_to_variables** (*dict, default=None*) – Dictionary describing the mapping between features (in X) and variables (in the model); it's optional because normally it has already been provided when the class has been instantiated.
- **optimizer** (*string, default="bfgs"*) –

  The optimizer that is going to be used. Acceptable values:

  - "bfgs", default: It's the Broyden-Fletcher-Goldfarb-Shanno algorithm, suitable for function with whose derivative can be computed. Generally faster, but might not always work.
  - "cma": Covariance-Matrix-Adaptation Evolution Strategy, derivative-free optimization. Much slower, but generally more effective than "bfgs".
- **optimizer_options** (*string, default=None*) – Options that can be passed to the optimization algorithm. Shape and type depend on the choice made for 'optimizer'.
- **n_jobs** (*int, default=0*) – Option that can be passed to the optimization algorithm, number of jobs to be executed in parallel. Default is zero, avoids the use of multiprocessing. -1 uses all available CPUs.
- **verbose** (*bool, default=False*) – If True, prints internal output to screen.

Returns

Return type None.

predict(*X*, *map_variables_to_features=None*, *verbose=False*)

Once the model is trained, this function can be used to predict the value of the target feature for samples in X. It will fail if the model has not been trained (TODO is this the default scikit-learn behavior?)

Parameters

- **X** (*array-like or sparse matrix, shape(n_samples, n_features)*) – Samples.
- **map_features_to_variables** (*dict, optional*) – A mapping between variables and features can be specified, if for some reason a different mapping than the one provided during instantiation is needed for this new array. The default is None, and in that case the model will use the previously provided mapping.

Returns **C** – Returns predicted values.

Return type array, shape(n_samples)

to_string()

Returns **model_description** – A human-readable string describing the model.

Return type str

3.1.4 Module contents

class humanmodels.**HumanClassifier**(*logic_expression: Union[str, dict]*, *map_variables_to_features: dict*, *random_state=None*)

Bases: sklearn.base.BaseEstimator, sklearn.base.ClassifierMixin

Human-style classification, using a dictionary of rules to be evaluated as logic expressions (e.g. "x*2 + 4*z > 0"), to then associate samples to a class.

Builder for the class.

Parameters

- **logic_expression** (*str or dictionary {int: string}*) – A string (or dictionary of strings).
- **map_variables_to_features** (*dict*) – Dictionary containing the mapping between variables and features indexes (in datasets).
- **target_class** (*list of int, optional*) – If several logic expressions are specified, a list of target classes must be passed as an argument, in order for HumanClassification to behave as a one-vs-all classifier. The default is None.

Returns

Return type None.

check_parameters()

Check coherence of the parameters.

Returns

Return type None.

error_function(*parameter_values*, *parameter_names*, *X*, *y*)

Error function, to be optimized. The parameters in each expression given for each class are replaced by the candidate parameter values, and the predictions of the model are then compared against class labels, obtaining a cost function value depending on the results.

fit(*X*, *y*, *optimizer: str = 'cma'*, *optimizer_options: Optional[dict] = None*, *n_jobs: int = 0*, *verbose: bool = False*)

Fits the internal model to the data, using features in X and known values in y.

Parameters

- **X** (*array, shape(n_samples, n_features)*) – Training data
- **y** (*array, shape(n_samples, 1)*) – Training values for the target feature/variable
- **optimizer** (*string, default="cma"*) –

  The optimizer that is going to be used. Acceptable values:

  - "cma": Covariance-Matrix-Adaptation Evolution Strategy, derivative-free optimization.
- **optimizer_options** (*dict, default=None*) – Dictionary of options that can be passed to the optimization algorithm. Shape and type depend on the choice made for 'optimizer'.
- **n_jobs** (*int, default=0*) – Option that can be passed to the optimization algorithm, number of jobs to be executed in parallel. Default is zero, avoids the use of multiprocessing. -1 uses all available CPUs.

- **verbose** (*bool, default=False*) – If True, prints internal output to screen.

Returns

Return type Self.

parameters_to_string(*c*)

predict(*X*)

Predict the class labels for each sample in X

Parameters **X** (*array, shape(n_samples, n_features)*) – Array containing samples, for which class labels are going to be predicted.

Returns **C** – Prediction vector, with a class label for each sample.

Return type array, shape(n_samples)

to_string()

variables_to_string(*c*)

class humanmodels.**HumanRegressor**(*equation_string: str, map_variables_to_features: Optional[dict] = None, target_variable_string: Optional[str] = None, random_state=None*)

Bases: sklearn.base.BaseEstimator, sklearn.base.RegressorMixin

Human-designed regressor, initialized with a sympy-compatible text string describing an equation. Also needs a dictionary mapping the correspondance between the variables named in the equation and the features in X.

Builder for the class.

Parameters

- **equation_string** (*str*) –

  String containing the equation of the model. Examples:

  1. "y = 2*x + 4"
  2. "4*x_0 + 5*x_1 + 6*x_2"

  If a left-hand side variable is NOT provided (as in example #2), the optional target_variable parameter must be specified.

- **map_features_to_variables** (*dict*) – Maps the names (or integer indexes) of the features to the variables in the internal symbolic expression representing the model.
- **target_variable_string** (*str, optional*) – String containing the name of the target variable. It's not necessary to specify target_variable if the left-hand part of the equation has been provided in equation_string. The default is None.

Returns

Return type None.

check_parameters()

Checks internal parameters for consistency, before starting data fitting.

Returns

Return type None.

error_function(*parameter_values*, *X*, *y*, *verbose=False*)

Error function to be optimized. Inside the error function, the local sympy expression will have its parameters replaced with candidate values.

Returns **mse** – Mean squared error of the model's prediction with the candidate parameters, agains true values in 'y'.

Return type float

fit(*X*, *y*, *map_variables_to_features: Optional[dict] = None*, *optimizer_options: Optional[dict] = None*, *optimizer: str = 'bfgs'*, *n_jobs: int = 0*, *verbose: bool = False*)
Fits the internal model to the data, using features in X and known values in y.

Parameters

- **X** (*array, shape(n_samples, n_features)*) – Training data
- **y** (*array, shape(n_samples, 1)*) – Training values for the target feature/variable
- **map_features_to_variables** (*dict, default=None*) – Dictionary describing the mapping between features (in X) and variables (in the model); it's optional because normally it has already been provided when the class has been instantiated.
- **optimizer** (*string, default="bfgs"*) –

  The optimizer that is going to be used. Acceptable values:

  - "bfgs", default: It's the Broyden-Fletcher-Goldfarb-Shanno algorithm, suitable for function with whose derivative can be computed. Generally faster, but might not always work.
  - "cma": Covariance-Matrix-Adaptation Evolution Strategy, derivative-free optimization. Much slower, but generally more effective than "bfgs".
- **optimizer_options** (*string, default=None*) – Options that can be passed to the optimization algorithm. Shape and type depend on the choice made for 'optimizer'.
- **n_jobs** (*int, default=0*) – Option that can be passed to the optimization algorithm, number of jobs to be executed in parallel. Default is zero, avoids the use of multiprocessing. -1 uses all available CPUs.
- **verbose** (*bool, default=False*) – If True, prints internal output to screen.

Returns

Return type None.

predict(*X*, *map_variables_to_features=None*, *verbose=False*)
Once the model is trained, this function can be used to predict the value of the target feature for samples in X. It will fail if the model has not been trained (TODO is this the default scikit-learn behavior?)

Parameters

- **X** (*array-like or sparse matrix, shape(n_samples, n_features)*) – Samples.
- **map_features_to_variables** (*dict, optional*) – A mapping between variables and features can be specified, if for some reason a different mapping than the one provided during instantiation is needed for this new array. The default is None, and in that case the model will use the previously provided mapping.

Returns **C** – Returns predicted values.

Return type array, shape(n_samples)

to_string()

Returns **model_description** – A human-readable string describing the model.

Return type str

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX