RS-HDMR-GPR Code Manual

Version 0.1.3

Owen Ren, Mohamed Boussaidi Ali, Sergei Manzhos, Dmitry Voytsekhovsky

Last Updated: April 5, 2021

1 Installation Guide

The latest version of the code can be obtained from github. The code comes with a IPython jupyter notebook (examples.ipynb) allows one to run and test the examples provided in the paper.

The **rshdmrgpr** package is installed as a Python package. Please install the 64-bit Python version if you are using a Windows machine. We also recommend the user install anaconda distribution to work with virtual environments and conda virtual environments. The code can also be run in google collab, or any environment that supports jupyter notebooks. In this manual we provide a guide for windows installation because all the work done are on Windows. However, there should not be anything that prevents a user from using the package on other operating systems (OS).

1.1 Windows Installation

On windows command prompt (or Anaconda prompt), create a virtual environment and install the package in a directory of your choosing.

```
C:\Users\new_user>python -m env env_name
C:\Users\new_user>env_name\Scripts\activate.bat
(env_name) C:\Users\new_user>pip install git+https://github.com/owen-
   ren0003/rshdmrgpr.git
```

After the package is installed, the user should be able to use it in terminal or any Python IDE with the correct environment setup.

```
(env_name) C:\Users\new_user>python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
   :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>  from rshdmrgpr import *
```

Users should consult pip and creating virtualenv for more detailed instructions and commands.

1.2 Anaconda Setup

Once Anaconda is installed, create a conda environment:

```
conda create -n env_name python=3.6
```

Activate the environment, which in this case is called 'env_name':

```
conda activate env_name
```

Install the rshdmrgpr package inside the conda environment:

```
pip install git+https://github.com/owen-ren0003/rshdmrgpr.git
```

Install jupyter lab:

```
conda install -c conda-forge jupyterlab
```

Finally, start jupyterlab to starting working interactively in a jupyter notebook:

```
jupyter lab
```

2 Python API

The code consists of four helper functions, followed by two classes. For the helper functions, we only detail only two of the helper function because the other are there mainly for convenience purposes. The sequential_fitting function is used to perform sequential fits of RSHDMRGPR models, where each model is fitted successively to the difference of the output and the prediction. The kernel_matrices helper function is used to generate standard unit matrices for hdmr along with isotropic RBF (Radial Basis Function) kernels. For class RSHDMRGPR, the choice of component functions is determined by specifying matrices and performing right multiplication with the feature matrix.

The first class creates objects that represent rs-hdmr-gpr models and are used to train and predict on data. Using the trained models from the first class, the second class is utilized to impute missing values. As in the accompanying article, the second class can only impute rows with exactly one missing value.

2.1 Helper function kernel_matrices

rshdmrgpr.**load_data**(*dataset*)

This function is used to create the RBF kernels and the matrix input for HDMR.

Parameters

- **dataset:** str – Specifies which dataset to load. One of 'h2o', 'KED', 'financial'.

Returns

- pandas DataFrame – The desired dataset.

rshdmrgpr.**kernel_matrices**(*order, dim, kernel_function=RBF*)

This function is used to create the RBF kernels and the matrix input for HDMR.

Parameters

- **order:** int – The order of HDMR to use.
- **dim:** int – The dimension of the feature space.
- **kernel_function: any GPR kernel** – Any kernel for GPR. Default is RBF.
- ****kwargs: keyword arguments** – Arguments for the given GPR kernel.

Returns

- list **of 2D-**numpy Array – List of matrices, used to select the features to train the component functions. Has size (dim choose order).
- list **of** RBF **kernels** – List of RBF kernels to use for training component function. Has size (dim choose order).

rshdmrgpr.**sequential_fitting**(*x_train, y_train, models, **params*)

This function is used to create the RBF kernels and the matrix input for HDMR.

Parameters

- **x_train:** pandas DataFrame – The training set.
- **y_train: 1d - array** – The target vector.
- **models:** list **of RSHDMRGPR models** – The list of RSHDMRGPR models to fit.
- ****params: dict** – variable number of key word arguments. Used to specify arguments for RSHDMRGPR.

Returns

- **list of 2D-numpy Array** – List of matrices, used to select the features to train the component functions. Has size (dim choose order).
- **list of RBF kernels** – List of RBF kernels to use for training component function. Has size (dim choose order).

rshdmrgpr.**sequential_prediction**(*x_test, models*)

This function returns a list of prediction for each order of fit. The i-th (0-based index) element contains the sequential fits up to order i (i.e. the sum of the predictions from **model**$[j]$ for all $j = 0, 1, ..., i$).

Parameters

- **x_test:** pandas DataFrame – The data set to be predicted on.
- **models:** list **of RSHDMRGPR models** – Contains the list of RSHDMRGPR models to be fitted sequentially.

Returns

- **list of 1d-array** – The i-th element contains the sequential fits up to order i.

2.2 RSHDMRGPR class

This class is used to fit a single instance of an hdmr model. A single instance is the sum of component functions determined by the matrices argument (a list of numpy matrices) that determines the component functions via right multiplication with the feature matrix of a given dataset. To perform sequential fitting, we need a list of RSHDMRGPR models and fit each sequentially where successive models fit to the difference of the label and the sum of the previous model predictions. Please see the main article Section 1, equations 5 and 6 for details.

class rshdmrgpr.**RSHDMRGPR**(*matrices, kernels*)

Bases: object

This class initializes the RS-HDMR-GPR model.

Parameters

- **matrices:** list **of 2D** numpy Array – The matrices that define the HDMR component functions. Every matrix must have the same number of rows equaling the dimension of the feature space.
- **kernels:** list **of kernels** – The list of sklearn.guassian_process.kernels to use for each HDMR component function.

2.2.1 RSHDMRGPR class methods

RSHDMRGPR.**verbose_print**(*msg, end=None, on=True*)

Predicts the results using the trained model parameters.

Parameters

- **msg:** str – The message to print.
- **end:** **None or** str – Specifies the ending of a line.
- **on:** bool – Specifies whether to print or not.

Returns None

RSHDMRGPR.**train**(*self, x_train, y_train, alphas=1e-7, n_restarts=1, cycles=50, scale_down=(0.2, 2), optimizer=None, opt_every=5,use_columns=None, initializer='even', report_loss=False, verbose=0*)

Trains the component functions for the HDMR model.

Parameters

- **x_train:** pandas DataFrame – The DataFrame containing features.
- **y_train:** pandas Series – The Series containing the target values.
- **alphas:** int **or (**list **of** int**)** – The noise level to be set for the component function, which is each a GPR model. If int, the noise level is the set to this for all cycles. To set different noise levels for different cycles, a list of int must be specified.
- **scale_down:** tuple – Must be a tuple, say (s, e) of size two. Training predictions for each component function will be multiplied by $\min\left\{s+\frac{(1-s)ec}{T}, 1\right\}$ on cycle c, where T denotes the total number of cycles.
- **cycles:** int – The number of self consistent cycles to use for training.
- **optimizer:** str **or (**list **of** str**)** – Must be a GPR optimizer or list of such. Please see the sklearn documentation for GaussianProcessRegressor optimizer.
- **opt_every:** int – Specifies every (opt_every) cycles to apply optimizer. Default=1 (optimizer applied every cycle). No difference if optimizer is None.
- **use_columns:** list **of** bool **or None** – Specifies which column to use (True indicates use, false otherwise). If None (the default), all columns are used.
- **n_restarts:** int – Positive integer indicating the number of restarts on the optimizer.
- **initializer:** list **of** float **or 'even'** – Initializes the starting targets for each component function. If 'even', every target is equal to y_train / len(use_columns).
- **report_loss:** bool – If True, saves the loss from the prediction on the training set over cycles. Only RMSE (Root mean-squared error) is supported for now.
- **verbose:** int – Sets the various print details option during training. Takes on values 0, 1, or 2. Default is 0.

Returns self, pandas DataFrame – Returns only the trained instance of self if report_rmse=False, otherwise returns the trained instance of self and a DataFrame containing the rmse values.

RSHDMRGPR.**predict**(*test_data, return_std=False*)

Predicts the results using the trained model parameters.

Parameters

- **test_data:** pandas DataFrame – The DataFrame containing the features of the test set. Should have one less column than the data used for training.
- **return_std:** bool – If True, the sum of all the component functions standard-deviation of the predictive distribution at the query points is returned along with the mean.

Returns

- **y_pred: 1D-**numpy Array – Sum of the mean of predictive distribution at query points of all component functions.
- **y_std: 1D-**numpy Array – Sum of the standard deviation of predictive distribution at query points of all component functions. Only returned when return_std=True.

RSHDMRGPR.**get_models**()

Returns the trained HDMR component functions.

Parameters None

Returns list **of** GaussianProcessRegressor – List of GaussianProcessRegressor from the sklearn.gaussian_process library, their trained instances.

2.3 FirstOrderHDMRImpute class

class rshdmrgpr.**FirstOrderHDMRImpute**(*models, division=1000*)

Bases: object

This class is initialized by vectorizing the 1D HDMR component functions and creating a dictionary of output values with division subdivisions. This is done because we need the inverse relation of each component function. This acts as a lookup table for inverse values where the endpoints of the subdivisions are inverse values. The larger the value of division, the more precise the inverse will be at approximating.

Parameters

- **models:** list of GaussianProcessRegressor – list of trained GaussianProcessRegressor models which represents the hdmr component functions of first order. Must be of first order (each has 1 input and 1 output).
- **division:** int – The number of divisions in the lookup table

2.3.1 FirstOrderHDMRImpute class methods

FirstOrderHDMRImpute.**get_table**()

Returns the lookup table for the component functions.

Parameters

Returns **panda** DataFrame – The lookup table for the HDMR component functions.

FirstOrderHDMRImpute.**get_yi**(*df_na*)

Modifies the DataFrame to contain the outputs of the first-order hdmr component functions. The output columns are concatenated past the last column of df_na.

Parameters

- **df_na:** pandas DataFrame – The DataFrame to impute. Must contain the output column and that column must be the last column.

Returns list – list of column indices for each row with missing df_na each corresponding to missing value column.

FirstOrderHDMRImpute.**impute**(*df_na, get_candidates=False, threshold=0.001*)

This function imputes the missing values given the input. Every single input row is expected to have at most one missing value.

Parameters

- **df_na:** pandas DataFrame – The DataFrame to impute. It is expected to contain the columns corresponding to 1D hdmr outputs (i.e. as an output of the get_yi function).
- **get_candidates:** bool – If True, returns all the candidates for imputation.
- **threshold:** float – The threshold distance to set for selecting candidates from look-up table. Correspond to δ in the paper (Section 2.2).

Returns

1) pandas DataFrame – The imputed DataFrame df_na if get_candidates=False.
2) pandas DataFrame, pandas Index, pandas Series, list **of** float – The imputed DataFrame df_na, the index of rows with null entries, the column names (indexed by the index of null entries) containing the null entry, and a list of candidates for imputing that missing value if get_candidates=True.

3 Algorithms

This section explains the implementation in pseudo-code format. We cover three algorithms mentioned in the main text: The RS-HDMR-GPR algorithm, the aglorithm on sequentially fitting the RS-HDMR-GPR models, and the imputation using 1D-HDMR. Notations:

1. m – number of entries for the data.
2. n – number of features the data contains.
3. $\alpha(c)$ – A scale down function as a function of the cycle number c.
4. $P_\ell(i)$ – A permutation of $\{1, 2, \ldots, \ell\}$ applied to element i.
5. X denotes a $m \times n$ matrix containing data corresponding to the features.
6. y denotes a $m \times 1$ vector containing the target data.

Algorithm 1: RS-HDMR-GPR

Input:

1. ℓ matrices $A_1, A_2, \ldots, A_\ell$, each of size $n \times k$.
2. GPR kernels K_i for $i = 1, 2 \ldots, \ell$. (Usually successive orders of HDMR).
3. The feature matrix X and label vector y.
4. A scale–down function $\alpha(c)$.

Output: Trained RS-HDMR-GPR model

Initialize vectors $\vec{y}_i = \frac{1}{\ell}\vec{y}$ and GPR models $\mathbf{model}_i = \varnothing$ for all $i = 1, 2, \ldots, \ell$.
Set P_ℓ to be the identity permutation. **for** $1 \leq c \leq$ # cycles **do**
  for $1 \leq i \leq \ell$ **do**
    1. Train $\mathbf{model}_{P(i)}$ using kernel $K_{P(i)}$ with input $XA_{P(i)}$ and output $\vec{y} - \sum_{j \neq P(i)} \vec{y}_j$.
    2. Update $\vec{y}_{P(i)} := (\mathbf{model}_{P(i)}$'s prediction on $XA_{P(i)}) \times \alpha(c)$
    3. Update $K_{P(i)}$ parameters to trained $\mathbf{model}_{P(i)}$'s new length scale if optimizer is applied.
  Choose a random permutation P_ℓ if permutation option is choosen.

Algorithm 2: Sequential Fitting

Input:

1. A list of untrained RS-HDMR-GPR models: $\mathbf{model}_i$ for $i = 1, 2, \ldots, k$.
2. The feature matrix X and label vector y.

Output: Trained instances of $\mathbf{model}_i$ for $i = 1, 2, \ldots, k$.

for $1 \leq i \leq k$ **do**
  Train $\mathbf{model}_i$ with respect to X and $\vec{y}$
  update $\vec{y}$ to $\vec{y} - \vec{z}$ where $\vec{z}$ is the prediction of $\mathbf{model}_i$ on X.

Algorithm 3: First order RS-HDMR-GPR imputation

Input:

1. 1D-HDMR component functions $f_1, f_2, \ldots, f_n$
2. A $m \times n$ feature matrix X whose columns entries are scaled to be between $[0, 1]$. For each row of X, there is at most one entry that is missing.
3. An integer $d > 0$.

Output: A $m \times n$ feature matrix X' that has it's missing values imputed

Compute the lookup table for the f_i's with d subdivisions of $[0, 1]$.
for *each row r of X with index i having a missing value* **do**
  Compute $y_i = \vec{y}_i - \sum_{j \neq i} y_j$
  Determine a list of input values in the lookup table closest to or within $\frac{2}{d}$ of y_i.
  The first element from this list is used for imputing the missing entry.

4 rshdmrgpr Examples

The supplementary IPython jupyter notebooks (examples.ipynb and physics_example.ipynb) for this manual demonstrates in detail how to use this package on the built-in datasets. These functions are also used on several examples. All of the examples presented in the accompanying document are worked out in these notebooks.