|  |  |
| --- | --- |
| Framework name | Function |
| Classification\_Advanced\_AutoHPODART | Fully automated (NAS + HPO) pipeline for bio-sequence classification with advanced functionality – sequence + hand-crafted features/numeric values |
| Classification\_Advanced\_semi\_manual | Partially automated (NAS) pipeline for bio-sequence classification with advanced functionality – sequence + hand-crafted features/numeric values |
| Classification\_Modelrepro\_advanced | Model reproduction template for above two |
| Classification\_Simple\_AutoHPODART | Fully automated (NAS + HPO) pipeline for bio-sequence classification with sequences only |
| Classification\_Simple\_semi\_manual\_learning | Partially automated (NAS) pipeline for bio-sequence classification with sequences only |
| Classification\_Modelrepro\_Simple | Model reproduction template for above two |
| Regression\_Advanced\_AutoHPODART | Fully automated (NAS + HPO) pipeline for bio-sequence regression with advanced functionality – sequence + hand-crafted features/numeric values |
| Regression\_Advanced\_semi\_manual | Partially automated (NAS) pipeline for bio-sequence regression with sequence + hand-crafted features/numeric values |
| Regression\_Modelrepro\_advanced | Model reproduction template for above two |
| Regression\_Simple\_AutoHPODART | Fully automated (NAS + HPO) pipeline for bio-sequence regression |
| Regression\_Simple\_semi\_manual\_learning | Partially automated (NAS) pipeline for bio-sequence regression |
| Regression\_Modelrepro\_Simple | Model reproduction template for above two |
| Standard\_protein\_structure\_based\_prediction | Fully automated (NAS + HPO) pipeline for standard dataset protein structure classification |
| Custom\_protein\_structure\_based\_prediction | Fully automated (NAS + HPO) pipeline for custom dataset protein structure classification |
|  |  |

**Simple: AutoHPODART :**

|  |  |  |
| --- | --- | --- |
| **Classification** | | |
| Modules to import:  from MEDHA.AutoTool.Classification.Simple.HPO\_DART import HPO\_DART | | |
| **HPO\_DART** | | |
| Lines of code:  myHpoObject = HPO\_DART(  sample\_data=sampledata,  in\_channel=1,  kernel=[1,3,5],  outchannel=1,  dataSet=whole\_Dataset,  lossfun='bce',  batch\_size=5,  acc\_thresold=65,  space={  'out\_channel\_input': hp.choice('out\_channel\_input',[50,100,125]),  'out\_channel\_f': hp.choice('out\_channel\_f',[25,50]),  'actfun': hp.choice('actfun',["ReLU6", "ReLU",'LeakyReLU']),  'drop': hp.uniform('drop', 0.0,0.3),  'UnitFCN\_vars': hp.choice('UnitFCN\_vars',[25,50]),  'nLayers\_vars': hp.choice('nLayers\_vars', [ 1,2]),  'loop': hp.choice('loop', [1,2]),  'num\_epochDART': hp.choice('num\_epochDART', [3,5])},  threshold=2,  predtype='binary',  L2lambdaDart=0.05)  avg\_val\_loss,avg\_valid\_acc,avg\_train\_acc,avg\_train\_loss,modelfinal,space,ParameterList = myHpoObject.Calling\_HPO\_DART(max\_evals=5,stoppage=3) | | |
| Output | Meaning | |
| avg\_val\_loss | Average validation losses of all folds combined for the best model found | |
| avg\_valid\_acc | Average validation accuracy of all folds combined for the best model found | |
| avg\_train\_acc | Average train accuracy of all folds combined for the best model found | |
| avg\_train\_loss | Average train loss of all folds combined for the best model found | |
| **modelfinal** | Final model obtained from the best model found used for independent test set prediction. | |
| space | Search-space containing these parameters: 'out\_channel\_input': 'out\_channel\_f','actfun','drop','UnitFCN\_vars','nLayers\_vars', 'loop':,'num\_epochDART'. | |
| ParameterList | This is the list of all hyperparameters used, as is useful to later reproduce the results. | |
| **Testing Independent test dataset** | | |
| **Same as Simple:Semi-Manual Learning Classification** | | |
|  | | |
| **Regression** | | |
| Modules to import:  from MEDHA.AutoTool.Regression.Simple.HPO\_DART import HPO\_DART | | |
| **HPO\_DART** | | |
| Lines of code:  myHpoObject = HPO\_DART(  sample\_data=sampledata,  in\_channel=1,  kernel=[1,3,5],  outchannel=1,  dataSet=whole\_Dataset,  lossfun='mse',  batch\_size=1000,  acc\_thresold=1,  space= {  'out\_channel\_input': hp.choice('out\_channel\_input',[25,100,125]),  'out\_channel\_f': hp.choice('out\_channel\_f',[25,50]),  'actfun': hp.choice('actfun',["ReLU6", "ReLU"]),  'drop': hp.uniform('drop', 0.1,0.3),  'UnitFCN\_vars': hp.choice('UnitFCN\_vars',[25,50]),  'nLayers\_vars': hp.uniform('nLayers\_vars', 1,2),  'loop': hp.uniform('loop', 1,2),  'num\_epochDART': hp.uniform('num\_epochDART',3,5)  },)  loss,modelfinal,space,ParameterList,pearsoncorrArr,spearmancorrArr,R\_squareArr = myHpoObject.Calling\_HPO\_DART(max\_evals=4,stoppage=3) | | |
| Output | | Meaning |
| loss | | Average validation loss of the best model found |
| **modelfinal** | | Final model obtained from the best model found used for independent test set prediction. |
| space | | Search-space containing these parameters: 'out\_channel\_input': 'out\_channel\_f','actfun','drop','UnitFCN\_vars','nLayers\_vars', 'loop':,'num\_epochDART'. |
| ParameterList | | This is the list of all hyperparameters used, as is useful to later reproduce the results. |
| pearsoncorrArr | | Pearson correlation obtained from the best model found |
| spearmancorrArr | | Spearman correlation obtained from the best model found |
| R\_squareArr | | R\_Square value obtained from the best model found |
| **Testing Independent test dataset** | | |
| **Same as Simple: Semi-Manual Learning Regression** | | |

**Explanation of each parameters in each sub-modules:**

|  |  |  |
| --- | --- | --- |
| **HPO\_DART (Classification)** | | |
| **The parameters of HPO\_DART constructor.** | | |
| Parameters | Explanation | Default available (Y/N) |
| sample\_data | This is mandatory input, output of DataProcess unit. | N |
| in\_channel | Number of Input channels. | N |
| kernel | Kernel sizes to be considered(A list). | Y |
| outchannel | This is the final out-channel, which indicates the number of predictions (1 for classification and can be one or more for regression depending on how many targets it is predicting).This is mandatory. | N |
| dataSet | This is mandatory input, which is output of DataProcess unit. This is the input torch-dataset. | N |
| lossfun | Loss function is mandatory to distinguish between binary and multi-label classification. | N |
| batch\_size | Batch size considered. | Y |
| acc\_thresold | This is the threshold that gets compared with DARTS accuracy(classification) during the process (Step 4,7 of the algorithm). | Y |
| pool\_size | Pooling size considered for all blocks | Y |
| drop | Dropout of every CNN | Y |
| space={  'out\_channel\_input': hp.choice('out\_channel\_input',[50,100,125]),  'out\_channel\_f': hp.choice('out\_channel\_f',[25,50]),  'actfun': hp.choice('actfun',["ReLU6", "ReLU",'LeakyReLU']),  'UnitFCN\_vars': hp.choice('UnitFCN\_vars',[25,50]),  'nLayers\_vars': hp.choice('nLayers\_vars', [ 1,2]),  'loop': hp.choice('loop', [1,2]),  'num\_epochDART': hp.choice('num\_epochDART', [3,5,7])}, | The search space which Hyperopt will optimize. Users can adjust the range of the values, keeping parameters unchanged. | Y |
| threshold | Threshold to consider during accuracy determination in DARTS (classification only). | Y |
| predtype | Prediction type (can be ‘binary’ for binary classification or ‘multi-label’ for multi-label classification).This is also mandatory input. | N |
| optimizerset | Optimizer used | Y |
| learning\_rate | Learning Rate | Y |
| L2lambdaDart | L2-Lambda | Y |
| momentumDart | Momentum | Y |
|  |  |  |
| **Parameters of the method: Calling\_HPO\_DART.** | | |
| Parameters | Explanation | Default available (Y/N) |
| max\_evals | Number of trials | N |
| stoppage | The stoppage criteria. If the loss does not change for the given number of times (say stoppage = 3), then the process would stop. | N |
| **HPO\_DART (Regression)** | | |
| Parameters | Explanation | Default available (Y/N) |
| sample\_data | This is mandatory input, output of DataProcess unit. | N |
| in\_channel | Number of Input channels. | N |
| kernel | Kernel sizes to be considered(A list). | Y |
| outchannel | This is the final out-channel, which indicates the number of predictions (can be one or more for regression depending on how many targets it is predicting).This is mandatory. | N |
| dataSet | This is mandatory input, which is output of DataProcess unit. This is the input torch-dataset. | N |
| lossfun | Loss function is mandatory to distinguish between KL-Div and MSE loss. | N |
| batch\_size | Batch size considered. | Y |
| acc\_thresold | This is the threshold that gets compared with DARTS loss(regression) during the process (Step 4,7 of the algorithm). | Y |
| pool\_size | Pooling size considered for all blocks | Y |
| space= {  'out\_channel\_input': hp.choice('out\_channel\_input',[25,50,75,100,125]),  'out\_channel\_f': hp.choice('out\_channel\_f',[25,50,60]),  'actfun': hp.choice('actfun',["ReLU6", "ReLU"]),  'drop': hp.uniform('drop', 0.1,0.3),  'UnitFCN\_vars': hp.choice('UnitFCN\_vars',[25,50,65]),  'nLayers\_vars': hp.uniform('nLayers\_vars', 1,3),  'loop': hp.uniform('loop', 1,2),  'num\_epochDART': hp.uniform('num\_epochDART',3,10)  } | The search space which Hyperopt will optimize. Users can adjust the range of the values, keeping parameters unchanged. | Y |
| optimizerset | Optimizer used | Y |
| learning\_rate | Learning Rate | Y |
| L2lambdaDart | L2-Lambda | Y |
| momentumDart | Momentum | Y |

*Although during object instantiation only 5 and for the second function call all 2 are mandatory user-input parameters*, yet it is recommended to users to specify their own acc\_threshold. Other parameters including the search-space has default values, set after multiple experimentations. However those can be adjusted as well, to optimize the model-performance.

**Advanced: Semi-Manual Learning :**

|  |  |  |
| --- | --- | --- |
| **Classification** | | |
| Modules to import:  from MEDHA.AutoTool.Classification.Advanced.SemiManualDART\_train import SemiManualDart\_train | | |
| **DartCaller** | | |
| Lines of code:  DartObject = SemiManualDart\_train()  modelfinal,exported\_arch,nas\_modules,ParameterList,DARTacc = DartObject.**DartCaller**(out\_channel\_input = 50,  out\_channel\_f = 75,  drop = 0.4,  UnitFCN\_vars = 65,  nLayers\_vars = 1,  loop = 1,  pool\_size = 1,  actfun = 'ReLU',  num\_epochs=7,  OptimizerDart = 'Adam',  sample\_data = sampledata,  in\_channel = 1,  kernel = [1,3,5],  batch\_size=5,  outchannel = 1,  chooseblocks=['block1','block2','block3','block4','block5','block6','block7'],  learning\_rateDart = 0.00018,  L2lambdaDart = 0.03 ,  momentumDart = 0.0,  dart\_dataset = whole\_Dataset,  lossfuntype = 'bce',  threshold=2) | | |
| Output | Meaning | |
| **modelfinal** | The model learnt by DART. DARTS finds best architecture and simultaneously trains the weights. So this is a sub-optimally trained model. This would be needed in further. | |
| exported\_arch | This gives the list of final chosen parameters by DART. It is in form of a dictionary with keys being some identifiers that translates to each operations/blocks. | |
| nas\_modules | The user can get to know the meaning of the keys in above parameter by looking into this. | |
| ParameterList | This is the list of all hyperparameters used, as is useful to later reproduce the results. | |
| DARTacc | This is the mean accuracy of the training (created by DartCaller) achieved by DART. | |
| **Concater** | | |
| Lines of code:  usenet,Concat\_dataset = DartObject.**Concater**(whole\_Dataset = whole\_Dataset,  **model** = modelfinal,  concatflag = True,  condata = condata,  modeltypeflag = 'dnn',  out\_param = 1,  nUnits=100,  nLayers=1,  createlist=ParameterList  ) | | |
| Output | | Meaning |
| usenet | | Gives the model (dnn/lstm) as per choice to be trained by K-Fold cross validation further |
| Concat\_dataset | | The concatenated dataset which will be inputted in next function. |
| **KFoldCrossValidator** | | |
| Lines of code:  train\_loss\_all,train\_acc,validation\_acc,avg\_train\_loss,avg\_train\_acc,avg\_val\_acc,bestmodel,avg\_val\_loss = DartObject.**KFoldCrossValidator**(k=3,  **crossvalidator\_dataset**=Concat\_dataset,  batch\_size=5,  **model**=usenet,  learning\_rate = 0.006,  L2lambda = 0.00002 ,  momentum = 0.0,  OptimizerKfold = 'Adam',  lossfuntype='bce',  num\_epochs=5,  predtype='binary') | | |
| Output | Meaning | |
| train\_loss\_all | Training loss of all epochs for all folds in a 2D-list. | |
| train\_acc | Training accuracy of all epochs for all folds in a 2D-list. | |
| validation\_acc | Validation accuracy of all epochs for all folds in a 2D-list. | |
| avg\_train\_loss | Average training loss for all folds combined. | |
| avg\_train\_acc | Average training accuracy for all folds combined. | |
| avg\_val\_acc | Average validation accuracy for all folds combined | |
| avg\_val\_loss | Average validation loss for all folds combined | |
| **bestmodel** | Model achieved after training the network , in this phase. This model is to be used for testing on independent set along with the DART-derived model in earlier step. | |
| **Testing Independent test dataset** | | |
| Modules to import:  from MEDHA.AutoTool.Classification.Advanced.SemiManualDart\_test import SemiManualDart\_test | | |
| Lines of code:  testobject = SemiManualDart\_test()  accuracy,precision,recall,f1\_score = testobject.predict( test\_loaderF,  modeldart = modelfinal,  modelkfold = bestmodel,  concatflagT = True,  condataT = condataT,  indexT = indexofdata,  labelsT = labelsT,  resulttype = 'bceranking',  indtruestart = 0,  indtrueend = 176  ) | | |
| Output | Meaning | |
| accuracy | Accuracy achieved on the test dataset | |
| precision | Precision achieved on the test dataset | |
| recall | Recall achieved on the test dataset | |
| f1\_score | F1-score achieved on the test dataset | |
| MCC (for binary only) | MCC achieved on the test dataset | |
| **Other outputs outputted in csv/text** | | |
| Output | Meaning | |
| out.csv/ Predictions\_binarylabel\_out.txt/ Predictions\_multilabel\_out.txt | This outputs either ranked predictions (out.csv) or just the predictions | |
| **Regression** | | |
| Modules to import:  from MEDHA.AutoTool.Regression.Advanced.SemiManualDART\_train import SemiManualDart\_train | | |
| **DartCaller** | | |
| Lines of code:  DartObject = SemiManualDart\_train()  modelfinal,exported\_arch,nas\_modules,ParameterList,DARTacc = DartObject.DartCaller(out\_channel\_input = 100,  out\_channel\_f = 25,  drop = 0.2,  UnitFCN\_vars = 50,  nLayers\_vars = 1,  loop = 1,  pool\_size = 1,  actfun = 'ReLU',  num\_epochs=5,  OptimizerDart = 'SGD',  sample\_data = sampledata,  in\_channel = 1,  kernel = [1,3,5],  batch\_size=10,  outchannel = 1,  chooseblocks=['block1','block2','block3','block4','block5','block6','block7'],  learning\_rateDart = 0.006,  L2lambdaDart = 0.00002 ,  momentumDart = 0.6,  dart\_dataset = whole\_Dataset,  lossfuntype = 'mse') | | |
| Output | | |
| **Same as Advanced (Classification)** | | |
| **Concater** | | |
| usenet,Concat\_dataset = DartObject.Concater(whole\_Dataset = whole\_Dataset,  model = modelfinal,  concatflag = True,  condata = condata,  modeltypeflag = 'dnn',  out\_param = 1,  losstype='mse',  createlist=ParameterList  ) | | |
| Output | | |
| **Same as Advanced (Classification)** | | |
| **KFoldCrossValidator** | | |
| Lines of code:  train\_loss\_all,test\_loss\_all,avg\_train\_loss,avg\_test\_loss,bestmodel,pearsoncorrArr,spearmancorrArr,R\_squareArr = DartObject.KFoldCrossValidator(usenet=usenet,#usenet is the model  k=5,  crossvalidator\_dataset=Concat\_dataset,  batch\_size=100,  lossfuntype='mse',  num\_epochs=3) | | |
| Output | Meaning | |
| train\_loss\_all | Train losses from all epochs for all folds | |
| test\_loss\_all | Validation losses from all epochs for all folds | |
| avg\_train\_loss | Average train losses of all folds combined | |
| avg\_test\_loss | Average validation losses of all folds combined | |
| **bestmodel** | Model achieved after training the network , in this phase. This model is to be used for testing on independent set along with the DART-derived model. | |
| pearsoncorrArr | Pearson correlation from all epochs for all folds .In case of multi-output predictions this is blank. | |
| spearmancorrArr | Spearman correlation from all epochs for all folds. In case of multi-output predictions this is blank. | |
| R\_squareArr | R\_square from all epochs for all folds. In case of multi-output predictions this is blank . | |
| **Testing Independent test dataset** | | |
| Modules to import:  from MEDHA.AutoTool.Regression.Advanced.SemiManualDart\_test import SemiManualDart\_test | | |
| Lines of code:  testobject = SemiManualDart\_test()  lossesKL = testobject.predict(rawdata=indexofdata,  labelnames=labelnames,  test\_loaderF=test\_loaderF,  **modeldart**=modelfinal,  **modelkfold**=bestmodel,  concatflagT=True,  condataT=condataT,  lossfun='div',  needinexcel='yes'  ) | | |
| Output | Meaning | |
| lossesKL | Mean loss from all the test data | |
| **Other outputs printed on screen and outputted in csv/text** | | |
| Output | Meaning | |
| Pearsoncorrelation | Net Pearsoncorrelation for single target predictions | |
| Spearmancorrelation | Net Spearmancorrelation for single target predictions | |
| R2 value | Net R2 value for single target predictions | |
| resultlist.txt | This is the net loss | |
| Storepredictions.txt | This stores all the predictions | |
| predictions.csv | This is outputted when ‘needinexcel’ is set to True. | |

**Explanation of each parameters in each sub-modules:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DartCaller** | | | | |
| Parameters | Explanation | | | Default available (Y/N) |
| out\_channel\_input | Output channel from the first input CNN-block | | | Y |
| out\_channel\_f | Output channel from the 7-mid level CNN-blocks | | | Y |
| drop | Dropout of every CNN | | | Y |
| UnitFCN\_vars | Number of units in the fully-connected layer | | | Y |
| nLayers\_vars | Number of layers in the fully-connected unit | | | Y |
| loop | Number of times the entire CNN-block is repeated (except input block) | | | Y |
| pool\_size | Pooling size considered for all blocks | | | Y |
| actfun | Activation function | | | Y |
| num\_epochs | Number of epochs | | | Y |
| OptimizerDart | Optimizer used | | | Y |
| sample\_data | This is mandatory input, output of DataProcess unit | | | N |
| in\_channel | Number of Input channels | | | Y |
| kernel | Kernel sizes to be considered(A list). | | | Y |
| batch\_size | Batch size considered | | | Y |
| outchannel | This is the final out-channel, which indicates the number of predictions (1 for classification and can be one or more for regression depending on how many targets it is predicting).This is mandatory. | | | N |
| chooseblocks | In this optional input, users can choose which blocks they would like to consider for their analysis from the 7-mid level blocks | | | Y |
| learning\_rateDart | Learning Rate | | | Y |
| L2lambdaDart | L2-Lambda | | | Y |
| momentumDart | Momentum | | | Y |
| dart\_dataset | This is mandatory input, which is output of DataProcess unit. This is the input torch-dataset to DartCaller. This is raw sequences only. | | | N |
| lossfuntype | Loss function is mandatory to distinguish between binary , multi-label classification and also regression loss type – KL-div or MSE. | | | N |
| **Parameter unique to Classification** | | | | |
| threshold | Threshold to consider during accuracy determination in DART(this parameter is for classification only). | | | Y |
| **Concater** | | | | |
| Parameters | | Explanation | Default available (Y/N) | |
| whole\_Dataset | | This is mandatory input, which is output of DataProcess unit. This is the input torch-dataset to DartCaller. This is raw sequences only. | N | |
| model | | Model derived from DartCaller | N | |
| concatflag | | This is set to true always when concatenation is needed. If user makes it False then only with the learnt tensor of raw sequences the process will proceed. *This user can use if they want to use LSTM with the raw sequences only*. | Y | |
| condata | | One needs to put the tensor form of handpicked features, which is the output of DataProcessing unit of Advanced category | N | |
| modeltypeflag | | The type of model, the user decides to choose: It can be MLP and should be abbreviated as ‘dnn’ or LSTM- abbreviated as ‘lstm’ | Y | |
| out\_param | | This is the final out-channel, which indicates the number of predictions (1 for classification and can be one or more for **regression** depending on how many targets it is predicting).This is mandatory. | N | |
| nUnits | | Number of units in each layer of DNN | Y | |
| nLayers | | Number of layers of DNN | Y | |
| nLSTMlayers | | Number of hidden layers of LSTM | Y | |
| n\_hiddenLSTM | | Number of units in each hidden layer of LSTM. | Y | |
| losstype | | Loss function is mandatory to distinguish between binary , multi-label classification and also regression loss type – KL-div or MSE. | N | |
| createlist | | User must input the ‘parameterlist’ which is achieved from the previous step. | N | |
| **KFoldCrossValidator** | | | | |
| Parameters | Explanation | | | Default available (Y/N) |
| k | Number of folds | | | Y |
| crossvalidator\_dataset | Concatenated dataset to be inputted (output of Concater) | | | N |
| batch\_size | Batch size | | | N |
| Model (classification)/usenet (regression) | This is the model derived from the previous step:Concater | | | N |
| learning\_rate | Learning Rate | | | Y |
| L2lambda | L2-Lambda | | | Y |
| momentum | Momentum | | | Y |
| OptimizerKfold | Optimizer used | | | Y |
| lossfuntype | Loss function type | | | Y |
| num\_epochs | Number of epochs | | | Y |
| **Parameter unique to Classification** | | | | |
| predtype | Prediction type – weather it is binary or multi-label | | | N |

|  |  |  |
| --- | --- | --- |
| **Testing Independent test dataset (classification)** | | |
| Parameters | Explanation | Default available (Y/N) |
| test\_loaderF | This is the output of the DataProcessingTest unit which contains the torch-dataset containing one-hot encoded raw sequences. | N |
| modeldart | Final model from DartCaller to be inputted here | N |
| modelkfold | Final model from KFoldCrossValidator to be inputted here | N |
| concatflagT | This is set to true always when concatenation is needed. This is present in the training model-Concater as well and should be of the same value. | Y |
| condataT | One needs to put the tensor form of handpicked features, which is the output of DataProcessingTest unit of Advanced category | N |
| indexT | Column in the test-csv which has data’s identifiers. | N |
| labelsT | We need to put the output:labelsT, achieved from DataPreprocessTest. These are tensor conversions of actual predictions which will help in model-evaluation | N |
| resulttype | MEDHA is equipped to rank the predictions, in case of binary from highest to lowest propensities with corresponding scores. It can also produce normal results, without ranks. However, the choice has to be inputted by the user. If we use, ‘bceranking’, then MEDHA ranks the predictions from highest to lowest scores, indicating the strength of each predictions. | N |
| indtruestart | In case we choose 'bceranking', we then need to arrange the outputs such that the positives are sorted before the negatives. In this parameter, the system takes in the **row number** from where the actual positive results start in the column of the test csv file. | N |
| indtrueend | In this parameter, the system takes in the **( row number- 1)** where the actual positive results end in the column of the test csv file. The tool is able to evaluate all metrics like accuracy, precision, recall, F1-score and MCC by calculating how many predictions are truly positive. | N |
| **Parameters of Testing Independent test dataset (regression)** | | |
| Parameters | Explanation | Default available (Y/N) |
| rawdata | Column in the test-csv which has data’s identifiers. (indexes) | N |
| labelnames | Labelnames (output of DataPreprocessTest) | N |
| test\_loaderF | This is the output of the DataProcessingTest unit which contains the torch-dataset containing one-hot encoded raw sequences. | N |
| modeldart | Final model from DartCaller to be inputted here | N |
| modelkfold | Final model from KFoldCrossValidator to be inputted here | N |
| concatflagT | This is set to true always when concatenation is needed. This is present in the training model-Concater as well and should be of the same value. | N |
| condataT | One needs to put the tensor form of handpicked features, which is the output of DataProcessingTest unit of Advanced category | N |
| lossfun | Loss function is mandatory to distinguish between KL-Div and MSE regression. | N |
| needinexcel | This is set to ‘yes’, when one needs to predict multi-column target in regression. | N |

Although there are many parameters, nevertheless most of them have default values specially for DartCaller , Concater and K-FoldCrossValidator while for predicting independent test dataset it becomes mandatory to input appropriate values , most of which are derived from earlier function-calls. *However, as earlier, with the python scripts we provided, users can directly use this tool on their task, changing fewer parameters and not needing to call each function as* ***complete frameworks have been provided in the scripts already***.

**Advanced: AutoHPODART :**

|  |  |  |
| --- | --- | --- |
| **Classification** | | |
| Modules to import:  from MEDHA.AutoTool.Classification.Advanced.HPO\_DART import HPO\_DART\_advanced | | |
| **HPO\_DART** | | |
| Lines of code:  myHpoObject = HPO\_DART\_advanced(  sample\_data = sampledata,  in\_channel = 1,  kernel = [1,3,5],  outchannel = 1,  dataSet = whole\_Dataset,  concatflag = True,  lossfun = 'bce',  batch\_size = 5,  modeltypeflag = 'dnn',  UnitFCN\_vars = 50,  nLayers\_vars = 1,  threshold = 2,  predtype = 'binary',  condata = condata,  acc\_thresold= 65)  avg\_val\_acc,avg\_val\_loss,avg\_train\_acc,avg\_train\_loss,modelDart,modelkfold,space,ParameterList = myHpoObject.Calling\_HPO\_DART(max\_evals=3,stoppage=2) | | |
| Output | Meaning | |
| avg\_val\_loss | Average validation losses of all folds combined for the best model found | |
| avg\_val\_acc | Average validation accuracy of all folds combined for the best model found | |
| avg\_train\_acc | Average train accuracy of all folds combined for the best model found | |
| avg\_train\_loss | Average train loss of all folds combined for the best model found | |
| modelDart | DART derived model. | |
| modelkfold | Model achieved from K-Fold cross validation. | |
| space | Search-space | |
| ParameterList | This is the list of all hyperparameters used, as is useful to later reproduce the results. | |
| **Testing Independent test dataset** | | |
| **Same as Advanced : Semi-Manual Learning Classification** | | |
|  | | |
| **Regression** | | |
| Modules to import:  from MEDHA.AutoTool.Regression.Advanced.HPO\_DART import HPO\_DART\_advanced | | |
| **HPO\_DART** | | |
| Lines of code:  myHpoObject = HPO\_DART\_advanced(  sample\_data = sampledata,  in\_channel = 1,  kernel = [1,3,5],  outchannel = 195,  dataSet = whole\_Dataset,  concatflag = True,  lossfun = 'div',  modeltypeflag = 'dnn',  batch\_size = 10,  condata = condata,  acc\_thresold = 2.5)  loss,modelDart,modelkfold,space,createlist,pearsoncorrArr,spearmancorrArr,R\_squareArr= myHpoObject.Calling\_HPO\_DART(max\_evals=6,stoppage=3) | | |
| Output | | Meaning |
| loss | | Average validation loss of the best model found |
| modelDart | | DART derived model. |
| modelkfold | | Model achieved from K-Fold cross validation. |
| space | | Search-space. |
| ParameterList | | This is the list of all hyperparameters used, as is useful to later reproduce the results. |
| pearsoncorrArr | | Pearson correlation obtained from the best model found |
| spearmancorrArr | | Spearman correlation obtained from the best model found |
| R\_squareArr | | R\_Square value obtained from the best model found |
| **Testing Independent test dataset** | | |
| **Same as Advanced: Semi-Manual Learning Regression** | | |

**Explanation of each parameters in each sub-modules:**

|  |  |  |
| --- | --- | --- |
| **HPO\_DART (Classification)** | | |
| **The parameters of HPO\_DART constructor.** | | |
| Parameters | Explanation | Default available (Y/N) |
| sample\_data | This is mandatory input, output of DataProcess unit. | N |
| in\_channel | Number of Input channels. | N |
| kernel | Kernel sizes to be considered(A list). | Y |
| outchannel | This is the final out-channel, which indicates the number of predictions (1 for classification and can be one or more for regression depending on how many targets it is predicting).This is mandatory. | N |
| dataSet | This is mandatory input, which is output of DataProcess unit. This is the input torch-dataset. | N |
| concatflag | This is set to true always when concatenation is needed. | Y |
| lossfun | Loss function is mandatory to distinguish between binary and multi-label classification. | N |
| batch\_size | Batch size considered. | Y |
| pool\_size | Pooling size considered for all blocks | Y |
| modeltypeflag | The type of model, the user decides to choose: It can be MLP and should be abbreviated as ‘dnn’ or LSTM- abbreviated as ‘lstm’ | Y |
| UnitFCN\_vars | Number of units in each hidden layer of the Fully-connected layer of the CNN-architecture | Y |
| nLayers\_vars | Number of hidden layers of the Fully-connected layer of the CNN-architecture | Y |
| loop | Number of loops for the CNN-architecture | Y |
| spacednn= {  'out\_channel\_input': hp.choice('out\_channel\_input',[25,50,75,100,125]),  'out\_channel\_f': hp.choice('out\_channel\_f',[25,50,75,100,125]),  'actfun': hp.choice('actfun',["ReLU6", "ReLU",'LeakyReLU']),  'drop': hp.uniform('drop', 0.3,0.5),  'unitsdnn': hp.choice('UnitFCN\_vars',[60,100,150,250,300]),  'layersdnn': hp.choice('nLayers\_vars', [ 1,2]),  'num\_epochDART': hp.choice('num\_epochDART', [3,5,7])  } | Search-space for the DNN (MLP) | Y |
| spacelstm= {  'out\_channel\_input': hp.choice('out\_channel\_input',[25,50,75,100,125]),  'out\_channel\_f': hp.choice('out\_channel\_f',[25,50,75,100,125]),  'actfun': hp.choice('actfun',["ReLU6", "ReLU",'LeakyReLU']),  'drop': hp.uniform('drop', 0.3,0.5),  'nLSTMlayers': hp.choice('nLSTMlayers',[1,2]),  'n\_hiddenLSTM': hp.choice('n\_hiddenLSTM', [10,50,100,150]),  'num\_epochDART': hp.choice('num\_epochDART', [3,5,7]),  } | Search-space for the LSTM | Y |
| threshold | Threshold to consider during accuracy determination in DARTS (classification only). | Y |
| predtype | Prediction type (can be ‘binary’ for binary classification or ‘multi-label’ for multi-label classification).This is also mandatory input. | N |
| optimizerset | Optimizer used | Y |
| learning\_rate | Learning Rate | Y |
| L2lambdaDart | L2-Lambda | Y |
| momentumDart | Momentum | Y |
| condata | One needs to put the tensor form of handpicked features, which is the output of DataProcessing unit of Advanced category | N |
| acc\_thresold | This is the threshold that gets compared with DARTS accuracy(classification) during the process (Step 4,7 of the algorithm). | Y |
| **Parameters of the method: Calling\_HPO\_DART.** | | |
| Parameters | Explanation | Default available (Y/N) |
| max\_evals | Number of trials | N |
| stoppage | The stoppage criteria. If the loss does not change for the given number of times (say stoppage = 3), then the process would stop. | N |
| **HPO\_DART (Regression)** | | |
| Parameters | Explanation | Default available (Y/N) |
| sample\_data | This is mandatory input, output of DataProcess unit. | N |
| in\_channel | Number of Input channels. | N |
| kernel | Kernel sizes to be considered(A list). | Y |
| outchannel | This is the final out-channel, which indicates the number of predictions (1 for classification and can be one or more for regression depending on how many targets it is predicting).This is mandatory. | N |
| dataSet | This is mandatory input, which is output of DataProcess unit. This is the input torch-dataset. | N |
| concatflag | This is set to true always when concatenation is needed. | Y |
| lossfun | Loss function is mandatory to distinguish between binary and multi-label classification. | N |
| batch\_size | Batch size considered. | Y |
| pool\_size | Pooling size considered for all blocks | Y |
| modeltypeflag | The type of model, the user decides to choose: It can be MLP and should be abbreviated as ‘dnn’ or LSTM- abbreviated as ‘lstm’ | Y |
| UnitFCN\_vars | Number of units in each hidden layer of the Fully-connected layer of the CNN-architecture | Y |
| nLayers\_vars | Number of hidden layers of the Fully-connected layer of the CNN-architecture | Y |
| loop | Number of loops for the CNN-architecture | Y |
| spacednn= {  'out\_channel\_input': hp.choice('out\_channel\_input',[25,50,75,100,125]),  'out\_channel\_f': hp.choice('out\_channel\_f',[25,50,75,100,125]),  'actfun': hp.choice('actfun',["ReLU6", "ReLU",'LeakyReLU']),  'drop': hp.uniform('drop', 0.3,0.5),  'unitsdnn': hp.choice('UnitFCN\_vars',[60,100,150,250,300]),  'layersdnn': hp.choice('nLayers\_vars', [ 1,2]),  'num\_epochDART': hp.choice('num\_epochDART', [3,5,7])  } | Search-space for the DNN (MLP) | Y |
| spacelstm= {  'out\_channel\_input': hp.choice('out\_channel\_input',[25,50,75,100,125]),  'out\_channel\_f': hp.choice('out\_channel\_f',[25,50,75,100,125]),  'actfun': hp.choice('actfun',["ReLU6", "ReLU",'LeakyReLU']),  'drop': hp.uniform('drop', 0.3,0.5),  'nLSTMlayers': hp.choice('nLSTMlayers',[1,2]),  'n\_hiddenLSTM': hp.choice('n\_hiddenLSTM', [10,50,100,150]),  'num\_epochDART': hp.choice('num\_epochDART', [3,5,7]),  } | Search-space for the LSTM | Y |
| optimizerset | Optimizer used | Y |
| learning\_rate | Learning Rate | Y |
| L2lambdaDart | L2-Lambda | Y |
| momentumDart | Momentum | Y |
| condata | One needs to put the tensor form of handpicked features, which is the output of DataProcessing unit of Advanced category | N |
| acc\_thresold | This is the threshold that gets compared with DARTS accuracy(classification) during the process (Step 4,7 of the algorithm). | Y |

Most of the parameters used above are defaults however users can adjust them for optimal tuning.

**ModelReproduction :**

|  |  |
| --- | --- |
| **Simple: Classification** | |
| **ModelReproduction** | |
| Modules to import:  from MEDHA.ModelReproduction.Classification.Simple.ModelReproductionSimpleClass import ModelReproduction | |
| Lines of code:  reproobj = ModelReproduction(parameterlist='ParameterList\_class\_simple\_01.pkl',  seqtype = 'proteinpadded',  mainmodel\_statedict='Best\_Semi\_Manual\_class\_simple\_01.pt',  numchannels=1,  max\_length\_of\_trainseq=350)  modelfinal = reproobj.GetModel() | |
| Output | Meaning |
| modelfinal | Pre-Saved Model placed on model-holder which can be used in testing |
| **Independent Test set testing** | |
| **Same as original process: contains two steps 1> DataProcessing 2> calling SemiManualDart\_test of Simple category** | |
| **Simple:Regression** | |
| Modules to import:  from MEDHA.ModelReproduction.Regression.Simple.ModelReproductionSimpleReg import ModelReproduction | |
| **The lines of code, outputs and process of calling test set is same as Simple: Classfication** | |

|  |  |
| --- | --- |
| **Advanced: Classification** | |
| **ModelReproduction** | |
| Modules to import:  from ModelReproduction.Classification.Advanced.ModelReproductionAdvancedClass import ModelReproduction | |
| Lines of code:  reproobj = ModelReproduction(parameterlist='ParameterList\_reg\_adv\_RBS\_sm.pkl',  seqtype = 'dna',  numchannels=1,  max\_length\_of\_trainseq=17,  statedict\_modelDart='Best\_adv\_reg\_RBS\_sm\_modeldart.pt',  statedict\_modelFold='Best\_adv\_reg\_RBS\_sm\_modelkfold.pt',)  modelfinal,use\_Net = reproobj.GetModel() | |
| Output | Meaning |
| modelfinal | Pre-Saved DART-Model placed on model-holder which can be used in testing |
| use\_Net | Pre-Saved K-Fold validated-Model placed on model-holder which can be used in testing |
| **Independent Test set testing** | |
| **Same as original process: contains two steps 1> DataProcessing 2> calling SemiManualDart\_test of Advanced category** | |
| **Advanced:Regression** | |
| Modules to import:  from ModelReproduction.Regression.Advanced.ModelReproductionAdvancedReg import ModelReproduction | |
| **The lines of code, outputs and process of calling test set is same as Advanced: Classfication** | |

**Explanation of each parameters in each sub-modules:**

|  |  |  |
| --- | --- | --- |
| **Simple: ModelReproduction (Classification and Regression)** | | |
| Parameters | Explanation | Default available (Y/N) |
| parameterlist | This input is provided by user. The parameter list which is also saved as outputted in every earlier processes | N |
| customalphabet | In case user used their own encoding then the same must be stated here | Y |
| customscheme | Custom encoding scheme (number of encoding characters) | Y |
| seqtype | Same as used in DataProcessing for the original runs. | N |
| mainmodel\_statedict | This is the saved model | N |
| numchannels | Number of channels | Y |
| max\_length\_of\_trainseq | This is the maximum length amongst all the sequences used in training data | N |
|  |  |  |
| **Advanced: ModelReproduction (Classification and Regression)** | | |
| Parameters | Explanation | Default available (Y/N) |
| parameterlist | This input is provided by user. The parameter list which is also saved as outputted in every earlier processes | N |
| customalphabet | In case user used their own encoding then the same must be stated here | Y |
| customscheme | Custom encoding scheme (number of encoding characters) | Y |
| seqtype | Same as used in DataProcessing for the original runs. | N |
| max\_length\_of\_trainseq | This is the maximum length amongst all the sequences used in training data | N |
| statedict\_modelDart | This is the saved DART-derived model | N |
| statedict\_modelFold | This is the saved K-Fold validated model | N |
| numchannels | Number of channels | Y |

**AutoMated Graph Learning for custom protein-structure dataset**

|  |  |  |
| --- | --- | --- |
| **Classification** | | |
| Modules to import:  import torch  from MEDHA.AutoToolGraph.Semimanualtrain import SemiManualDart\_train  from MEDHA.AutoToolGraph.crossvalidation import cross\_validation\_with\_val\_set  from MEDHA.AutoToolGraph.Graphein\_Caller import Graphein\_Caller | | |
| **Graphein\_Caller** | | |
| Lines of code:  Graphein\_object = Graphein\_Caller(datacsv='structural\_rearrangement\_data\_1final.csv', #name of the input file  name='protein\_2', #internal parameter for Graphein  batch\_size=5)  dataset,train\_loader,val\_loader,max\_nodes= Graphein\_object.GetData() | | |
| Output | Meaning | |
| dataset | Entire graph dataset (each graph is represented as adjacency matrix and feature matrix) | |
| train\_loader | Train\_laoder ,input to DartCaller | |
| val\_loader | Vlidation\_laoder ,input to DartCaller | |
| max\_nodes | Maximum number of nodes (max limit is set to 1000) | |
| **DartCaller** | | |
| Lines of code:  myHpoObject = HPO\_DART(  input\_channel=dataset.num\_node\_features, #input channel -> number of node features  outchannel=dataset.num\_classes, # number of output classes (e.g. binary = 2)  max\_nodes=max\_nodes,  percent\_dec=0.25,#uses DIFFPOOL so asks how much decrease of number of nodes needed  batch\_size=5,  OptimizerDart='Adam',  learning\_rateDart = 0.0001,  dataset=dataset,  acc\_thresold=50,  epochs=5,  intepochs=3,  R=1)  loss\_mean ,accsval, model,trainlossmeanf,createlist,space = myHpoObject.Call\_Process() | | |
| Output | | Meaning |
| model | | The model learnt by DART+HPO. |
| createlist | | Parameters used in model, can be used later for model reproduction. |
| space | | DARTS supernet config |
| trainlossmeanf | | Mean Training loss |
| accsval | | This is the mean accuracy of the test set |
| loss\_mean | | This is the mean loss of the test set |