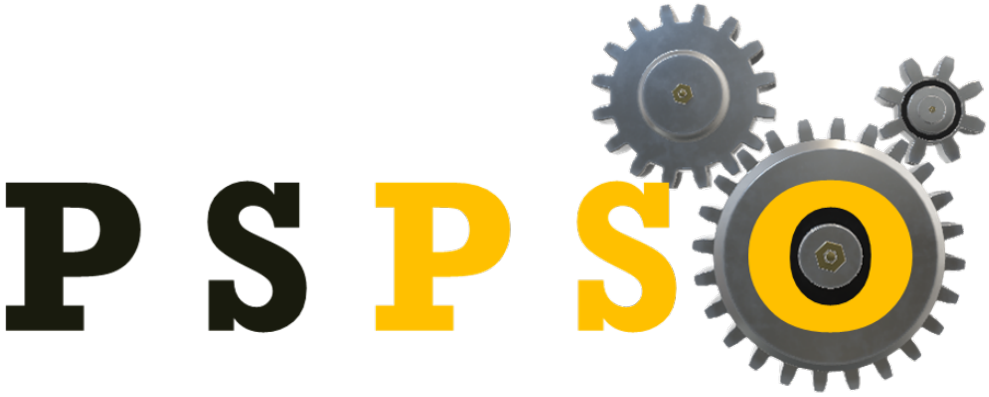
pspso

Release 0.0.7

Ali Haidar

Mar 20, 2020

CONTENTS


PSPSO

CHAPTER

ONE

OVERVIEW AND INSTALLATION

1.1 Overview

pspso is a python library for selecting machine learning algorithms parameters. The first version supports two single algorithms: Multi-Layer Perceptron (MLP) and Support Vector Machine (SVM). It supports two ensembles: Extreme Gradient Boosting (XGBoost) and Gradient Boosting Decision Trees (GBDT).

Two types of machine learning tasks are supported by pspso:

- Regression.
- Binary classification.

Three scores are supported in the first version of pspso:

- **Regression** :
  - Root Mean Square Error (RMSE) for regression tasks
- **Binary Classication** :
  - Area under the Curve (AUC) of the Receiver Operating Characteristic (ROC)
  - Accuracy

1.2 Installation

Use the package manager pip to install pspso.

```
pip install pspso
```

CHAPTER

TWO

USAGE

2.1 MLP Example

pspso is used to select the machine learning algorithms parameters. It is assumed that the user has already processed and prepared the training and validation datasets, which are usually used to build the model. Below is an example for using the pspso to select the parameters of the MLP. It should be noticed that pspso handles the MLP random weights intialization issue that may cause losing the best solution in consecutive iterations.

The following example demonstrates the selection process of the MLP parameters. The params variable details the parameters utilized for selection. The task and the score are defined as binary classification and score. Then, the PSO is used to select the parameters of the MLP. Results will be provided back to user. It should be mentioned that the number of neurons has been left as a default value and was not given for selection in this example.

```
from pspso import pspso
params = {"optimizer":['adam','nadam','sgd','adadelta'],
    "learning_rate":  [0.01,0.2,2],
    "hiddenactivation": ['sigmoid','tanh','relu'],
    "activation": ['sigmoid','tanh','relu']}
task='binary classification'
score='auc'
number_of_particles=4
number_of_iterations=5
p=pspso('mlp',params,task,score)
p.fitpspso(X_train,Y_train,X_val,Y_val,number_of_particles=number_of_particles,
               number_of_iterations=number_of_iterations)
p.printresults()
```

In this example, four parameters were examined: optimizer, learning_rate, hiddenactivation, and activation. The number of neurons in the hidden layer was kept as default.

2.2 XGBoost Example

The XGBoost is an implementation of boosting decision trees. It is assumed that at this stage the user has already prepared the training and validation cohorts. Five parameters were utilized for selection: objective, learning rate, maximum depth, number of estimators, and subsample. Three categorical values were selected for the objective parameter. The learning rate parameter values range between 0.01 and 0.2 with 2 decimal point, maximum depth ranges between 1 and 10 with 0 decimal points (1,2,3,4,5,6,7,8,9,10), etc. The task and score are selected as regression and RMSE respectively. The number of particles and number of iterations can be left as default values if needed. Then, a pspso instance is created. By applying the fitpspso function, the selection process is applied. Finally, results are printed back to the user. The best model, best parameters, score, time, and other details will be saved in the created instance for the user to check.

```
from pspso import pspso
params = {
                "objective":['reg:tweedie',"reg:linear","reg:gamma"],
        "learning_rate":  [0.01,0.2,2],
        "max_depth": [1,10,0],
        "n_estimators": [2,200,0],
        "subsample": [0.7,1,1]}
task="regression"
score="rmse"
number_of_particles=20
number_of_iterations=40
p=pspso('xgboost',params,task,score)
p.fitpspso(X_train,Y_train,X_val,Y_val,
                            number_of_particles=number_of_particles,
                            number_of_iterations=number_of_iterations)
print("PSO search:")
p.printresults()
```

2.3 User Input

The user enters the type of the algorithm ('mlp','svm','xgboost','gbdt'); the task type ('binary classification', 'regression'), score ('rmse','acc', or 'auc'). The user can keep the parameters variable empty, where a default set of parameters and ranges is loaded for each algorithm.

```
from pspso import pspso
task='binary classification'
score='auc'
p=pspso.pspso('xgboost',None,task,score)
```

Pspso allows the user to provide a range of parameters for exploration. The parameters vary between each algorithm. For this current version, up to 5 paramaters can be explored at the same time.

The parameters are encoded in json object that consists of *key,value* pairs:

```
params = {
        "objective":['reg:tweedie',"reg:linear","reg:gamma"],
        "learning_rate":  [0.01,0.2,2],
        "max_depth": [1,10,0],
        "n_estimators": [2,200,0],
        "subsample": [0.7,1,1]}
```

The key can be any parameter belonging to to the algorithm under investigation. The value is a list. Pspso will check the type of the first element in the list, which will determine if the values of the parameter are categorical or numerical.

Categorical Parameters

If the parameter values are *categorical*, string values are expected to be found in the list, as shown in *objective* parameter. The values in the list will be automatically mapped into a list of integers, where each integer represents a value in the original list. The order of the values inside the list affect the position of the value in the search space.

Numerical Parameters

On the other side, if the parameter is numerical, a list with three elements is expected [lb,ub, rv]:

- **lb**: repesents the lowest value in the search space
- **ub**: represents the maximum value in the search space

- **rv**: represents the number of decimal points the parameter values are rounded to before being added for training the algorithm

For e.g if you want pspso to select n_estimators, you add the following list [2,200,0] as in the example. By that, the lowest n_estimators will be 2, the highest to be examined is 200, and each possible value is rounded to an integer value (0 decimal points).

2.4 Other parameters

The user is given the chance to handle some of the default parameters such as the number of epochs in the MLP. The user can modify this by changing a pspso class instance. For e.g., to change the number of epochs from 50 to 10 for an MLP training:

```
from pspso import pspso
task='binary classification'
score='auc'
p=pspso.pspso('mlp',None,task,score)# in case of empty set of params (None) default␣
↪search space is loaded
p.defaultparams['epochs']=10
```

The verbosity can be modified for any algorithm, which allows showing details of the training process:

```
from pspso import pspso
task='binary classification'
score='auc'
p=pspso.pspso('mlp',None,task,score)
p.verbosity=1
```

Early stopping rounds for supporting algorithm can be modified, default is 60:

```
from pspso import pspso
task='binary classification'
score='auc'
p=pspso.pspso('xgboost',None,task,score)
p.early_stopping=10
```

CHAPTER

THREE

FUNCTIONS

3.1 ML Algorithms Functions

forward_prop_gbdt(particle, task, score, ...)	Train the GBDT after decoding the parameters in variable particle.
forward_prop_xgboost(particle, task, score, ...)	Train the XGBoost after decoding the parameters in variable particle.
forward_prop_svm(particle, task, score, ...)	Train the SVM after decoding the parameters in variable particle.
forward_prop_mlp(particle, task, score, ...)	Train the MLP after the decoding the parameters in variable particle.

3.2 Selection Functions

fitpspso([X_train, Y_train, X_val, Y_val, ...])	Select the algorithm parameters based on PSO.
fitpsgrid([X_train, Y_train, X_val, Y_val])	Select the algorithm parameters based on Grid search.
fitpsrandom([X_train, Y_train, X_val, ...])	Select the algorithm parameters based on radnom search.

The fitpsrandom() and fitpsgrid() were implmented as two default selection methods. With fit random search, the number of attempts to be tried is added by the user as a variable. In grid search, all the possible combinations are created and investigated by the package. These functions follow the same encoding schema used in fitpspso(), and were basically added for comparison.

3.3 Parameters Encoding/Decoding Functions

readparameters([params, estimator, task])	Read the parameters provided by the user.
decode_parameters(particle)	Decodes the parameters of a list into a meaningful set of parameters.

3.4 Other Functions

f(q, estimator, task, score, X_train, ...)	Higher-level method to do forward_prop in the whole swarm.
rebuildmodel(estimator, pos, task, score, ...)	Used to rebuild the model after selecting the parameters.
printresults()	Print the results found in the pspso instance.
calculatecombinations()	A function that will generate all the possible combinations in the search space.

CHAPTER

FOUR

MODULE SUMMARY

class pspso.**pspso**(*estimator='xgboost'*, *params=None*, *task='regression'*, *score='rmse'*)

This class searches for algorithm parameters by using the Particle Swarm Optimization (PSO) algorithm.

calculatecombinations()

A function that will generate all the possible combinations in the search space. Used mainly with grid search

Returns

combinations: list A list that contains all the possible combinations.

static decode_parameters(*particle*)

Decodes the parameters of a list into a meaningful set of parameters. To decode a particle, we need the following global variables:parameters, defaultparameters, paramdetails, and rounding.

static f(*q*, *estimator*, *task*, *score*, *X_train*, *Y_train*, *X_val*, *Y_val*)

Higher-level method to do forward_prop in the whole swarm.

Inputs

x: numpy.ndarray of shape (n_particles, dimensions) The swarm that will perform the search

Returns

numpy.ndarray of shape (n_particles,) The computed loss for each particle

fitpsgrid(*X_train=None*, *Y_train=None*, *X_val=None*, *Y_val=None*)

Select the algorithm parameters based on Grid search.

Grid search was implemented to match the training process with pspso and for comparison purposes. I have to traverse each value between x_min, x_max. Create a list seperating rounding value.

fitpspso(*X_train=None*, *Y_train=None*, *X_val=None*, *Y_val=None*, *number_of_particles=5*, *number_of_iterations=10*, *options={'c1': 1.49618*, *'c2': 1.49618*, *'w': 0.7298}*)

Select the algorithm parameters based on PSO.

Inputs

X_train: numpy.ndarray of shape (a,b) Contains the training input features, a is the number of samples, b is the number of features

Y_train: numpy.ndarray of shape (a,1) Contains the training target, a is the number of samples

X_train: numpy.ndarray of shape (c,b) Contains the validation input features, c is the number of samples, b is the number of features

Y_train: numpy.ndarray of shape (c,1) Contains the training target, c is the number of samples

number_of_particles: integer number of particles in the PSO search space.

number_of_iterations: integer number of iterations.

options: dictionary A key,value dict of PSO parameters c1,c2, and w

Returns

pos: list The encoded parameters of the best solution

cost: float The score of the best solution

duration: float The time taken to conduct random search.

model: The best model generated via random search

combinations: list of lists The combinations examined during random search

results: list The score of each combination in combinations list

fitpsrandom(*X_train=None*, *Y_train=None*, *X_val=None*, *Y_val=None*, *number_of_attempts=20*)
Select the algorithm parameters based on radnom search.

With Random search, the process is done for number of times specified by a parameter in the function.

Inputs

X_train: numpy.ndarray of shape (a,b) Contains the training input features, a is the number of samples, b is the number of features

Y_train: numpy.ndarray of shape (a,1) Contains the training target, a is the number of samples

X_train: numpy.ndarray of shape (c,b) Contains the validation input features, c is the number of samples, b is the number of features

Y_train: numpy.ndarray of shape (c,1) Contains the training target, c is the number of samples

number_of_attempts: integer The number of times random search to be tried.

Returns

pos: list The encoded parameters of the best solution

cost: float The score of the best solution

duration: float The time taken to conduct random search.

model: The best model generated via random search

combinations: list of lists The combinations examined during random search

results: list The score of each combination in combinations list

static forward_prop_gbdt(*particle*, *task*, *score*, *X_train*, *Y_train*, *X_val*, *Y_val*)
Train the GBDT after decoding the parameters in variable particle. The particle is decoded into parameters of the gbdt. Then, The gbdt is trained and the score is sent back to the fitness function.

Inputs

particle: list of values (n dimensions) A particle in the swarm

task: regression, binary classification, or binary classification r the task to be conducted

score: rmse (regression), auc (binary classification), acc (binary classification) the type of evaluation

X_train: numpy.ndarray of shape (m, n) Training dataset

Y_train: numpy.ndarray of shape (m,1) Training target

X_val: numpy.ndarray of shape (x, y) Validation dataset

Y_val: numpy.ndarray of shape (x,1) Validation target

Returns

variable, model the score of the trained algorithm over the validation dataset, trained model

static forward_prop_mlp(*particle*, *task*, *score*, *X_train*, *Y_train*, *X_val*, *Y_val*)
Train the MLP after the decoding the parameters in variable particle.

static forward_prop_svm(*particle*, *task*, *score*, *X_train*, *Y_train*, *X_val*, *Y_val*)
Train the SVM after decoding the parameters in variable particle.

static forward_prop_xgboost(*particle*, *task*, *score*, *X_train*, *Y_train*, *X_val*, *Y_val*)
Train the XGBoost after decoding the parameters in variable particle. The particle is decoded into parameters of the XGBoost. This function is similar to forward_prop_gbdt The gbdt is trained and the score is sent back to the fitness function.

Inputs

particle: list of values (n dimensions) A particle in the swarm

task: regression, binary classification, or binary classification r the task to be conducted

score: rmse (regression), auc (binary classification), acc (binary classification) the type of evaluation

X_train: numpy.ndarray of shape (m, n) Training dataset

Y_train: numpy.ndarray of shape (m,1) Training target

X_val: numpy.ndarray of shape (x, y) Validation dataset

Y_val: numpy.ndarray of shape (x,1) Validation target

Returns

variable, model the score of the trained algorithm over the validation dataset, trained model

printresults()
Print the results found in the pspso instance. Expected to print general details like estimator, task, selection type, number of attempts examined, total number of combinations, position of the best solution, score of the best solution, parameters, details about the pso algorithm.

static readparameters(*params=None*, *estimator=None*, *task=None*)
Read the parameters provided by the user.

Inputs

params: dictionary of key,values added by the user This dictionary determines the parameters and ranges of parameters the user wants to selection values from.

estimator: string value A string value that determines the estimator: 'mlp','xgboost','svm', or 'gbdt'

task: string value A string valeu that determines the task under consideration: 'regression' or 'binary classification'

Returns

parameters The parameters selected by the user

defaultparams Default parameters

x_min: list The lower bounds of the parameters search space

x_max: list The upper bounds of the parameters search space

rounding: list The rounding value in each dimension of the search space

bounds: dict A dictionary of the lower and upper bounds

dimensions: integer Dimensions of the search space

params: Dict Dict given by the author

static rebuildmodel(*estimator*, *pos*, *task*, *score*, *X_train*, *Y_train*, *X_val*, *Y_val*)
Used to rebuild the model after selecting the parameters.

CHAPTER

FIVE

FUTURE WORK

5.1 Additional Parameters

To add new parameters to the currently supported algorithms, two functions should be updated

The **read_params** function should include default details about the parameter, The **forward_prop_algorithmname** function should add the parameter to the initialization process

5.2 New Algorithms

Adding a new algorithm is more complex as you will be required to add an objective function that will detail the training and evaluation process.

New Optimizers Two main optimizers are currently supported. These algorithms are built in the pyswams function.

The default is globalbest pso, however the user can specify the local pso The pso parameters are set to default in each case and can be modified by the user.

5.3 Crossvalidation

We are working towards adding the cross validation support that will take the training data and number of folds, then split the records and train each fold. Finally, the average performance is retuned to the user.

5.4 Multi-Class Classification

We are also working on adding multi-class classification and data oversampling techniques.

CHAPTER

SIX

CONTRIBUTING

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change.

Please make sure to update tests as appropriate.

We are working towards adding the cross validation support that will take the training data and number of folds, then split the records and train each fold. Finally, the average performance is retuned to the user.

We are also working on adding multi-class classification and data oversampling techniques.

CHAPTER

SEVEN

LICENSE

INDEX