pspso

Release 0.0.5

Ali Haidar

Mar 17, 2020

CONTENTS:

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 can be used with 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

CHAPTER

ONE

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. Below is an example for using the pso 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.

```
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

Five parameters of the xgboost are searched and explored.

```
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)
```

(continues on next page)

(continued from previous page)

```
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

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 user can provide an empty set of parameters. By that, a default search space is created.

How are the parameters encoded ?

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 Training details

The user is given the chance to handle some of the default parameters such as the number of epochs. The user can modify this by changing a pspso class intance. For e.g., if you need 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, …)	Calculates the fitness value of the encoded parameters in variable particle.
forward_prop_xgboost(particle, task, score, …)	This function accepts the particle from the PSO fitness function.
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, …])	fitpso search
fitpsgrid([X_train, Y_train, X_val, Y_val])	Grid search was implemented to match the training process with pspso and for comparison purposes.
fitpsrandom([X_train, Y_train, X_val, …])	With Random search, the process is done for number of times specified by a parameter in the function.

CHAPTER

FOUR

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()

generate 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:

global variable parameters global variable defaultparameters global variable paramdetails global variable 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*)

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=2*, *number_of_iterations=2*, *options={'c1': 0.5*, *'c2': 0.3*, *'w': 0.4}*)

fitpso search

fitpsrandom(*X_train=None*, *Y_train=None*, *X_val=None*, *Y_val=None*, *number_of_attempts=20*)

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

static forward_prop_gbdt(*particle*, *task*, *score*, *X_train*, *Y_train*, *X_val*, *Y_val*)

Calculates the fitness value of the encoded 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*)
This function accepts the particle from the PSO fitness function. 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 results

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

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

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

INDEX