GuangAn-B 广安-B

Package for Tuning (3st Generation)

24/06/2023

Background

The purpose of this package is to provide a sophisticated framework for Divide and Conquer tuning. *The Criteria of Third Generation Tuning is to be able to work on bounded semi-continuous fields.*

The package takes in X and y data for train, validate and test as DataFrame, as well as a dictionary of (hyperparameters bounds-> string: hyperparameter lower/upper bounds as a list, or hyperparameter choices of ordinal/nominal hyperparameter values as a set).

Algorithm Description

GuangAn-B begins by tuning the vertices/corner combinations of the bounded field. It then progresses to the next 'round' whereby the centre 'combination' of this set of bounds (in this case, the original field boundary) is tuned – this is referred to as the 'centre combination'.

Unlike previous generations of tuners, the combinations haven't been generated in advance using set values of every hyperparameter that the user wants to be tested, rather each new centre is generated on the fly by taking the mean value of the previous bounds for every hyperparameter. In the case where the hyperparameter has discrete values, it splits the original field into many (the number being the same as its number of discrete values) other fields with one less dimension, where in those fields the values for this specific parameter is now constant (of course, for multiple discrete hyperparameters the field will be split into $n*m$ smaller fields, where n and m are the numbers of values in the two discrete hyperparameters respectively; the new fields will have k less dimensions than the original field, with k being the number of discrete hyperparameters that are being tuned). In the case where a hyperparameter is semi-continuous (i.e. ordinal values), its median rank value will be attempted to be taken, but if after several rounds there is only one value left in the semi-continuous/ordinal hyperparameter, it is treated as discrete.
Users need to specify whether a variable is categorical (discrete) after they have entered bounds.

At every 'round' except the first round, GuangAn observes whether the 'centre observation' has a validation score not less than the previous round's best score (note, not current best score – round best scores get updated at the end of each round) by more than 0.005 (i.e. *centre score - last round best score* >= *-0.005*), and if this condition is satisfied, then it continues to create new boundaries using the current boundary and the centre that has just been tuned, and repeat the process at the start of the paragraph. Else, this particular boundary does not need to be further divide-and-conquered as it is not increasing the validation score. Do keep in mind that many, many boundaries will be created as the divide and conquer process evolves.

The key difference between GuangAn and GuangAn-B is that it no longer uses Linear Regression to determine whether there is a good fit, but the mechanism of training and evaluating all the boundary points are nonetheless retained to ensure a wide range of combinations are trained, to give the algorithm the best chance of finding the field global maximum.

At the end of each round, only the new boundaries that were derived from the centre combinations with the top *max(64,* 2^d*)* validation scores are proceeded into the next round, as part of a pruning mechanism. This is necessary or else the number of new boundaries that are generated will expand exponentially as the 'rounds' grows higher, and most of the combinations being pursued are stuck at mediocre validation score regions.

The algorithm has a 'wall-time'-like mechanism in that if all continuous hyperparameter's lower bounds and upper bounds in a new boundary is smaller than 0.1, then that new boundary will not be pursued as it is considered too fined grained and not worthy of being investigated. Note that despite this, this algorithm will still reach into finer values than those of lower generation (i.e. n_estimators down to unit values, instead of coarse 3, 6, 9, 12, 24, 48 etc; subsample values finer than 0.1 level).

Note also that values that are trained as exponential values in lower generation algorithms are changed considered by their log(10) values, so for a fairer mean-value when tuning.
Users need to specify whether a variable needs to undergo transformation after they have set bounds.

If the observed 'centre combination' produces a negative validation score after the third 'round', it is automatically discarded as very likely the algorithm has searched into an area of the field with terrible combinations. There is also a protective mechanism that ensures every 'centre' will be pushed through in the first two rounds, to prevent situations where the tuning process is ended because the first two rounds' centres had low validations scores by randomness,

When the initial round by round 'guidance' stage has terminated (i.e. no new boundaries need to be tested), the boundary containing the maximum observed validation score will undergo a further round (just by itself), to ensure that there is no further maximum within the boundary. If a new maximum is found, then the 'boundaries containing the maximum' will be adjusted, and go through the 'cruise' process again, until the maximum observed validation score is unchanged after a round.

Class

Class	Purpose
`GuangAn`	Object that performs divide and conquer tuning

Methods:

Methods	Purpose
GuangAn()	Initialisation
`read_in_data(train_x, train_y, val_x, val_y, test_x, test_y)`	Read in Train Test Split data Parameters: train_data – pd.DataFrame val_data - pd.DataFrame test_data - pd.DataFrame
`read_in_model(model, type)`	Read in the underlying model class that we want to tune to get optimal parameters for Parameters: model – any model **class** that allows .fit() and .predict() type – str – either "Classification" or "Regression"
`set_hyperparameters(parameter_ranges_orig)`	Read in the different values of each hyperparameters we want to try. Function will automatically generate each combination Parameters: parameter_ranges_orig – dict of str:list/str:set/str:dict– str is hyperparameter name (strictly as defined in model class), and list has 2 values in ascending order denoting bounds; set contains values in ascending order.

`set_non_tuneable_hyperparameters(non_tuneable_hyperparameter_choice)`	Reads in values for non-tuneable hyperparameters (i.e. doesn't need to clog up the tuning output csv) Parameters: non_tuneable_hyperparameter_choices – dict of str:int
`read_in_transform(transform_update)`	Specifies which tuneable hyperparameter values should be log10 transformed during tuning. Parameters: transform_update – dict of str:str – key str is hyperparameter name (strictly as defined in model class), and value str is type of transform *currently only support '10^' which represents log10 transformation
`read_in_categorical(categorical_update)`	Specifies which tuneable hyperparameter values should be considered categorical during tuning Parameters: categorical_update – list of strs - str is hyperparameter name (strictly as defined in model class)
`set_features(ningxiang_output)`	Reads in feature combinations for tuning Parameters: ningxiang_output – dict of tuple:float
`set_tuning_result_saving_address(address)`	Set saving address for tuning output csv

	Parameters: address – str – does not need to include '.csv'
`set_best_model_saving_address(address)`	Set best model saving address Parameters: address – str – does not need to include '.pickle'
`tune()`	Begin tuning process
`read_in_tuning_result_df(df_address, object_address)`	Read in existing dictionary from .pickle consisting of tuning result. Automatically populates result array and checked array if csv columns match parameter choices **even though users are reading in a pickled dictionary, the original method name was still retained in line with original JiaXing methods.* Parameters: df_address – str – include '.csv' object_address – str – include '.pickle
`view_best_combo_and_score()`	View the current best combination and its validation score

Objects:

Objects	Purpose
`train_x`	DataFrame
`train_y`	Series
`val_x`	DataFrame
`val_y`	Series
`test_x`	DataFrame
`test_y`	Series
`tuning_result`	DataFrame
`model`	model **class**
`parameter_ranges`	Dictionary -dict of str:list/str:set/str:dict– str is hyperparameter name (strictly as defined in model class), and list has 2 values in ascending order denoting bounds; set contains values in ascending order.
`transform`	Dictionary dict of str:str – key str is hyperparameter name (strictly as defined in model class), and value str is type of transform
`categorical`	Dictionary str:bool – str is hyperparameter name (strictly as defined in model class), and bool indicates whether a hyperparameter should be considered categorical
`hyperparameters`	list
`checked_dict`	dict
`tuning_result_saving_address`	str
`best_model_saving_address`	str
`best_score = -np.inf`	int
`best_combo`	list

`best_clf`	model **object**
`clf_type`	str – 'Regression' or 'Classification'

`regression_extra_output_columns = [`
`'Train r2',`
`'Val r2',`
`'Test r2',`
`'Train RMSE',`
`'Val RMSE',`
`'Test RMSE',`
`'Train MAPE',`
`'Val MAPE',`
`'Test MAPE',`
`'Time']`
	List (pre-setted)

`classification_extra_output_columns =`
`[`
`'Train accu',`
`'Val accu',`
`'Test accu',`
`'Train balanced_accu',`
`'Val balanced_accu',`
`'Test balanced_accu',`
`'Train f1',`
`'Val f1',`
`'Test f1',`
`'Train precision',`
`'Val precision',`
`'Test precision',`
`'Train recall',`
`'Val recall',`
`'Test recall',`
`'Time']`
	list (pre-setted)

Dependencies

pandas

numpy

sklearn

statsmodels