YangZhou-B 扬州-B

Package for Tuning (2nd Generation)

11/04/2023

Background

YangZhou, the native city of Chinese Paramount Leader Jiang Zemin (江泽民), is what this Second Generation Tuning Package is named after. The purpose of this package is to provide a sophisticated framework for Greedy tuning. *The Criteria of Second Generation Tuning is to not have to train every specified discrete combination to get the optimum discrete result – or close to the optimum.*

The package takes in X and y data for train, validate and test as DataFrame, as well as a dictionary of {hyperparameters name -> string: hyperparameter values as a list}, and autogenerates all combinations of these hyperparameters to be tuned.

Algorithm Description

YangZhou-B begins by train-searching (i.e. searching the field by training the combination) all *Cruise Combinations* (mathematical combinations of all *cruise indices* from each dimension: *cruise indices* being i.e. [0, 4], [0, 5], [0, 3, 6] or [0, 4, 7] for dimensions containing 5, 6, 7 and 8 values respectively. The maximum gap between two indices is 5, minimum is 3).

Then, starting with the median combination (median index of each dimension) as the initial core, the *Guidance Algorithm* is activated, in which the all the horizontal/vertical neighbouring combinations are searched (i.e. all the combinations which is same as the core except for one dimension being +1 or -1 compared to previously). If score(neighbour)-score(core)>= -0.005, then the the neighbour is added as the new core.

The *Guidance Algorithm* is then repeated for each of the new cores. When no new cores need to be tested, the maximum scoring combination have all surrounding neighbours searched, and if a new maximum scoring combination is found, then it will also get its neighbours searched until no new maximum scoring combination can be found. The *Guidance Algorithm* is then terminated.

The *Cruise algorithm* is then subsequently activated, in which each of the cruise combinations scores will be compared to the current best scoring combination and its surrounding +1/-1 neighbour block (including itself). If a cruise combination's score is higher than the

$$warning\ threshold = \ mean(best\ surrounding\ block) - qt(0.95) * \frac{sd}{\sqrt{3^d}}$$

then the *Guidance Algorithm* will be restarted on that Cruise combination.

The *Cruise Algorithm* terminates once all cruise combinations have been compared to the warning threshold (which could change as the *Cruise Algorithm* goes on)

Once the *Cruise Algorithm* ends, the *Guidance Algorithm* gets activated one more time starting at the current maximum scoring combination, and the whole *YangZhou-B Algorithm* ends when this call of *Guidance Algorithm* is finished.

Note: although scores of certain combinations will undoubtedly be called upon multiple times, they can be stored and thus the expensive basic operation of train-searching a combination will only ever need to be completed once for each combination.

Algorithm Assumptions

1. The scores observed from the same {data, model, hyperparameter combination, split size} belongs to the same underlying population which are normally distributed around a theoretical value.

    i.e. Accuracies of SVM on a fixed set of hyperparameters with 80-20 split size on the same set of data (but with random holdouts of 80% training data) is considered to be sampled from the same population (and thereby same distribution)

Class

Class	Purpose
YangZhouB	Object that performs greedy tuning

Methods:

Methods	Purpose
YangZhouB()	Initialisation
read_in_data(train_x, train_y, val_x, val_y, test_x, test_y)	Read in Train Test Split data Parameters: train_x – pd.DataFrame train_y - pd.Series val_x - pd.DataFrame val_y - pd.Series test_x - pd.DataFrame test_y – pd.Series
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_choices)	Read in the different values of each hyperparameters we want to try. Function will automatically generate each combination Parameters: parameter_choices – dict of str:list – str is hyperparameter

	name (strictly as defined in model class), and list is sorted values of hyperparameter which we want to try out.
`set_non_tuneable_hyperparameters(non_tuneable_hyperp arameter_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
`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'
`tune(key_stats_only = False)`	Begin tuning process If key_stats_only = True then don't calculate non important stats Parameters: key_stats_only – bool
`tune_parallel(part, splits, key_stats_only = False)`	Begin tuning process, splitting all combinations into

	splits parts and tune the *part*-th part (of Cruise). If key_stats_only = True then don't calculate non important stats Parameters: key_stats_only – bool
`read_in_tuning_result_df(address)`	Read in existing DataFrame from .csv consisting of tuning result. Automatically populates result array and checked array if csv columns match parameter choices Parameters: address – str – include '.csv'
`set_tuning_best_model_saving_address( address)`	Set address for exporting best model as a pickle Parameters: address – str – does not need to 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_choices	Dictionary -str:list – str is hyperparameter name (strictly as defined in model class), and list is sorted values of hyperparameter which we want to try out.
hyperparameters	list
feature_n_ningxiang_score_dict	Dictionary -str:float – str is hyperparameter name (strictly as defined in model class), and float is its NingXiang score
non_tuneable_parameter_choices	Dictionary -str:str/float/int - str is hyperparameter name (strictly as defined in model class), and values are valid hyperparameter values for model
checked	np.array
result	np.array
checked_core	np.array

	value = 1: appended onto list of cores to be checked
	value = 2: actually checked as a core
`been_cruised`	np.array value = 1: been checked as core, so don't need to be appended as a cruise value = 2: actually checked as a cruise combo
`been_best`	np.array
`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'
`n_items`	list - denoting how many values in each hyperparameter dimensions

`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',`
	list (pre-setted)

'Val precision', 'Test precision', 'Train recall', 'Val recall', 'Test recall', 'Time']	

Dependencies

pandas

numpy

scipy

sklearn

Test Result

1. Time

    YangZhou-B's algorithm will undoubtedly take more time than JiXi on top of the required time for tuning; but from testing, the maximum time required to run YangZhou-B on a dataset modelled on real data was 2.07 seconds on Google Colab, which is approximately the time to train one combination for the average model.

    Thus, YangZhou-B should be a time saver considering the amount of hyperparameter combinations it doesn't need to tune, especially if each hyperparameter combination takes a long time to tune.

2. Accuracy

Batch	Percentage of test cases when Algorithm output == Actual Max	Percentage of test cases Algorithm output >= Actual Max – 0.005
1	93.35%	99.88%
2	86.15%	96.20%
3	88.33%	100.00%
7	77.50%	95.00%

Batch	Algorithm output == Actual Max	Algorithm output >= Actual Max – 0.005
Real (4)	78.26%	91.30%

    The maximum difference between algorithm output and actual max in batch 4 (real data) was 0.0007.

3. Percentage of Hyperparameter Combinations searched

Batch	Mean	Median	Max
1	15.45%	5.87%	1%

2	12.51%	5.71%	73.47%
3	8.48%	1.35%	65.71%
7	10.60%	1.46%	83.33%

Batch	Mean	Median	Max
Real (4)	23.29%	19.11%	52.92%

On average, YangZhou-B only tunes less than 25% of all designated hyperparameter combinations.

Test Result (Interact)

1. Time

   YangZhouB's algorithm will undoubtedly take more time than JiXi on top of the required time for tuning; but from testing, the maximum time required to run YangZhouB on a dataset modelled on real data was 13.5435 seconds on Google Colab, which is approximately the time to train one combination for the average model.

   Thus, YangZhouB should be a time saver considering the amount of hyperparameter combinations it doesn't need to tune, especially if each hyperparameter combination takes a long time to tune.

2. Accuracy

Batch (Interact)	Percentage of test cases when Algorithm output == Actual Max	Percentage of test cases Algorithm output >= Actual Max – 0.005
1	92.71%	97.74%
2	88.33%	95.83%

Batch	Algorithm output == Actual Max	Algorithm output >= Actual Max – 0.005
Real (3)	78.26%	91.30%

   The maximum difference between algorithm output and actual max in batch 3 (real data) was 0.0007.

3. Percentage of Hyperparameter Combinations searched

Batch (Interact)	Mean	Median	Max
1	21.75%	11.28%	100%
2	14.22%	5.99%	66.67%

Batch	Mean	Median	Max
Real (3)	23.29%	19.11%	52.92%

On average, YangZhouB only tunes less than 25% of all designated hyperparameter combinations.