JiaoCheng 交城

Package for Feature-by-Feature Tuning

11/04/2023

Background

JiaoCheng, the native city of Chinese Transitional-Period Paramount Leader Hua Guofeng (华国峰), is what this Feature-by-Feature Tuning Package is named after. The purpose of this package is to provide a framework for feature-by-feature tuning, a different (and in most cases faster but less accurate) method compared to JiXi, YangZhou etc.

Sometimes, a data scientist would be stuck in the midst of data cleaning, but would like to get a glimpse of how well this data is currently performing as a benchmark, and hence does not need to necessarily find the global maximum in the field space. JiaoCheng, being more greedy and hence training less combinations and taking less time than JiXi, is suitable for this purpose, and is fittingly named after Hua's home town as Hua oversaw China's governance during its transitional period after the death of Mao Zedong.

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}, dictionary of default values for each hyperparameter and list of order of features, and autogenerates all combinations of these hyperparameters to be tuned.

JiaoCheng starts at the default values combination, and searches through different values of first hyperparameter whilst holding other hyperparameter values constant. The maximum combination from this search gets updated as the new 'default value combination' (now called 'current max combination') and the second hyperparameter is searched through holding other hyperparameter values of the 'current max combination' fixed. Once all hyperparameter have been searched through in this manner, if the 'current max combination' is the same as that before this round of all hyperparameter being searched, then the algorithm is terminated. Else, another round of search is undertaken.

The idea was taken from the Gibbs Sampling Algorithm in statistics.

Class

Class	Purpose
`JiaoCheng`	Object that performs feature-by-feature tuning

Methods:

Methods	Purpose
`JiaoCheng()`	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_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
`set_features(ningxiang_output)`	Reads in feature combinations for tuning Parameters: ningxiang_output – dict of tuple:float
`set_tuning_order(order)`	Sets the order of tuning for hyperparameters in JiaoCheng tuning Parameters: order – list
`set_hyperparameter_default_values(default_values)`	Sets the default values for hyperparameters in JiaoCheng tuning Parameters: default_values – dict of str:int/float/str
`set_tuning_result_saving_address(address)`	Set saving address for tuning output csv Parameters: address – str – does not need to include '.csv'
`change_tuning_style(type, seed = None, outer_most_layer = 2, randomise = True)`	Set which type of tuning order to use. 'a': as if nested (according to order of dictionary input to set_hyperparameters()) 'b': (reset to 'a') before random shuffle using inputted seed, or default seed 19421221

	‘c’: (reset to a) before setting to layer by layer order ‘d’: (reset to a) (reset to c) before setting to diag-hor -> layer by layer. Automatically randomised by default seed Parameters: type – str – ‘a’ or ‘b’ or ‘c’ or ‘d’ seed – int – for ‘b’ and ‘c’ outer_most_layer – the outer most layer for ‘c’ and ‘d’ to actually order for, before remaining are all random randomise – bool – whether or not to randomise ‘c’
`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
`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
`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’
`combos`	List of lists
`n_items`	list - denoting how many values in each hyperparameter dimensions
`hyperparameter_tuning_order`	list of hyperparameters

`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`

Test Result (Interact)

1. Time

   JiaoCheng'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 JiaoCheng 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, JiaoCheng 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

   Note: Tuning order and Default Values makes a difference for JiaoCheng! All tests on JiaoCheng assumed numerical order of features and default value = first value

<u>Batch</u> (Interact)	<u>Percentage of test cases when Algorithm output == Actual Max</u>	<u>Percentage of test cases Algorithm output >= Actual Max – 0.005</u>
1	79.69%	92.71%
2	84.17%	94.17%

<u>Batch</u>	<u>Algorithm output == Actual Max</u>	<u>Algorithm output >= Actual Max – 0.005</u>
Real (3)	82%	96%

3. Percentage of Hyperparameter Combinations searched

<u>Batch</u> (Interact)	<u>Mean</u>	<u>Median</u>	<u>Max</u>
1	14.82%	5.92%	76%
2	12.12%	3.76%	61.9%

Batch	Mean	Median	Max
Real (3)	6.7%	5.08%	20.99%

On average, JiaoCheng only tunes less than 7% of all designated hyperparameter combinations.