



JiaoCheng 交城

Package for Feature-by-Feature Tuning

24/06/2023

Background

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.

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

<u>Class</u>	<u>Purpose</u>
JiaoCheng	Object that performs feature-by-feature tuning

Methods:

<u>Methods</u>	<u>Purpose</u>
<i>JiaoCheng()</i>	Initialisation
<code>read_in_data(train_x, train_y, val_x, val_y, test_x, test_y)</code>	<p>Read in Train Test Split data</p> <p>Parameters:</p> <p><code>train_x</code> – <code>pd.DataFrame</code> <code>train_y</code> - <code>pd.Series</code> <code>val_x</code> - <code>pd.DataFrame</code> <code>val_y</code> - <code>pd.Series</code> <code>test_x</code> - <code>pd.DataFrame</code> <code>test_y</code> – <code>pd.Series</code></p>
<code>read_in_model(model, type)</code>	<p>Read in the underlying model class that we want to tune to get optimal parameters for</p> <p>Parameters:</p> <p><code>model</code> – any model class that allows <code>.fit()</code> and <code>.predict()</code></p> <p><code>type</code> – str – either “Classification” or “Regression”</p>
<code>set_hyperparameters(parameter_choices)</code>	<p>Read in the different values of each hyperparameters we want to try. Function will automatically generate each combination</p> <p>Parameters:</p> <p><code>parameter_choices</code> – 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.</p>

set_non_tuneable_hyperparameters(non_tuneable_hyperparameter_choice)	<p>Reads in values for non-tuneable hyperparameters (i.e. doesn't need to clog up the tuning output csv)</p> <p>Parameters: non_tuneable_hyperparameter_choices – dict of str:int</p>
set_features(ningxiang_output)	<p>Reads in feature combinations for tuning</p> <p>Parameters: ningxiang_output – dict of tuple:float</p>
set_tuning_order(order)	<p>Sets the order of tuning for hyperparameters in JiaoCheng tuning</p> <p>Parameters: order – list</p>
set_hyperparameter_default_values(default_values)	<p>Sets the default values for hyperparameters in JiaoCheng tuning</p> <p>Parameters: default_values – dict of str:int/float/str</p>
set_tuning_result_saving_address(address)	<p>Set saving address for tuning output csv</p> <p>Parameters: address – str – does not need to include '.csv'</p>
tune(key_stats_only = False)	<p>Begin tuning process</p> <p>If key_stats_only = True then don't calculate non important stats</p> <p>Parameters: key_stats_only – bool</p>
read_in_tuning_result_df(address)	<p>Read in existing DataFrame from .csv consisting of tuning result.</p>

	<p>Automatically populates result array and checked array if csv columns match parameter choices</p> <p>Parameters: address – str – include ‘.csv’</p>
set_tuning_best_model_saving_address(address)	<p>Set address for exporting best model as a pickle</p> <p>Parameters: address – str – does not need to include ‘.pickle’</p>
view_best_combo_and_score()	<p>View the current best combination and its validation score</p>

Objects:

<u>Objects</u>	<u>Purpose</u>
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</u> <u>when Algorithm output ==</u> <u>Actual Max</u>	<u>Percentage of test cases</u> <u>Algorithm output >=</u> <u>Actual Max – 0.005</u>
1	79.69%	92.71%
2	84.17%	94.17%

<u>Batch</u>	<u>Algorithm output ==</u> <u>Actual Max</u>	<u>Algorithm output >=</u> <u>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%

<u>Batch</u>	<u>Mean</u>	<u>Median</u>	<u>Max</u>
Real (3)	6.7%	5.08%	20.99%

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