NingXiang 宁乡

Package for Discrete Feature Combinations Ordering

24/06/2023

Background

The purpose of this package is to provide an ordering of discrete feature combinations for tuning packages JiXi and YangZhou so that feature combinations can participate in tuning like a hyperparameter in a meaningful way.

The naming was particularly chosen as Liu Shaoqi was the principal theorist of the First Generation of Leaders under Paramount Leader Mao Zedong, whose native city alongside those of other paramount leaders are used to perform tuning, and Discrete Feature Combination Ordering is highly similar to the theoretical work that must be undergone before tuning can begin.

The package performs feature combination ordering by linear regression sqrt(r^2) and random forest regressor feature importance, which represents the multi-feature vs label equivalent of pearson’s coefficient and NMI respectively.

Algorithm Description

Linear Regression sqrt(r^2)

-Starting with no features in the combination, iteratively add the feature that will increase r^2 of the resulting y~X linear model using training features by the most

Then, {combo: NingXiang score} will be {X, sqrt(r^2 of y~X)}

Random Forest feature importance

-First use `RandomForest (n_estimators = 100, max_depth = 12, max_features = 0.75, random_state = self._seed, ccp_alpha = 0, max_samples = 0.75)` to build model based on training data

-Then, using this model's feature importance, iteratively add features in reverse importance, and setting NingXiang score as the sum of this set of features' importance

Note:
-Under both use cases NingXiang score will be between [0, 1], however the linear regression use case is not guaranteed to reach 1.
-Classification models should only use Random Forest feature importance

Class

Class	Purpose
`NingXiang`	Object that performs discrete feature combinations ordering

Methods:

Methods	Purpose
`NingXiang()`	Initialisation
`read_in_train_data(train_x, train_y, val_x = None, val_y = None)`	Read in data Parameters: train_x – pd.DataFrame train_y - pd.Series val_x – pd.DataFrame val_y - pd.Series
`set_model_type(type)`	Read in the type of model that we are trying to build Parameters: type – str – either "Classification" or "Regression"
`get_lr_based_feature_combinations(min_features = 0, gap = 1)`	Builds lr model and gets NingXiang score based on sqrt of r^2, iteratively adding feature that increases r^2 by the most each time. Can set number of features in minimum feature combo (i.e. avoid having combo of just 1 or 2 features because some models can't train with too few features) Can set the gap between number of features in neighbouring combinations (i.e. for Natural Language Processing there are too many features to try all)

	Parameters: min_features – int gap - int
`get_rf_based_feature_combinations(min_features = 0, gap = 1, n_jobs = 1)`	Builds rf model and gets NingXiang output based on feature importance. Can set number of features in minimum feature combo (i.e. avoid having combo of just 1 or 2 features because some models can't train with too few features) Can set the gap between number of features in neighbouring combinations (i.e. for Natural Language Processing there are too many features to try all) Parameters: min_features – int gap – int n_jobs - int
`get_rf_based_feature_combinations_from_feature_importance(feature_importance = None, min_features = 0, gap = 1)`	Uses ready made feature importance to create NingXiang output Can set number of features in minimum feature combo (i.e. avoid having combo of just 1 or 2 features because some models can't train with too few features)

	Can set the gap between number of features in neighbouring combinations (i.e. for Natural Language Processing there are too many features to try all) Parameters: feature_importance – dict – str:float min_features – int gap - int
`show_rf_stats()`	Display the rf feature importance dataframe, and also the validation score (if validation score were inputted in first place)
`export_ningxiang_output(address)`	Export current NingXiang output object as a pickle object Parameters: address – str – does not need to include '.pickle'

Objects:

Objects	Purpose
`train_x`	DataFrame
`train_y`	Series
`clf_type`	str – ‘Regression’ or ‘Classification’
`ningxiang_output`	dict – tuple:float
`object_saving_address`	str

Dependencies

`pandas`

`numpy`

`sklearn`