ControlBurn Documentation

version v0.0.1

Brian Liu

June 27, 2022

Contents

Welcome to ControlBurn’s documentation!

ControlBurnModel.py

This module contains functions to build and burn forests and select hyperparameters.

class ControlBurn.ControlBurnModel.ControlBurnClassifier (`alpha`=0.1, `max_depth`=10, `build_forest_method`='bagboost', `polish_method`=RandomForestClassifier(max_features='sqrt'), `custom_forest`=[])

alpha
: Regularization hyperparameter.

  Type: float

max_depth
: Maximum depth of trees to grow in the ensemble.

  Type: float

threshold
: Convergence threshold.

  Type: float

tail
: Number of iterations to check convergence on.

  Type: int

forest
: List of DecisionTreeRegressor objects. Represents the full ensemble.

  Type: list

subforest
: List of DecisionTreeRegressor objects. Represents the sparse ensemble.

  Type: list

features_selected_
: List of features selected by the subforest.

  Type: list

feature_importances_
: List of feature importance scores computed from the subforest.

  Type: list

weights
: Solution coefficients to the LASSO optimization problem.

  Type: list

alpha_range
: Range of alphas enumerated when computing the regularization path.

coef_path
: LASSO solution coefficients obtained by computing the regularization path.

polish_method
: Method used to polish the final model on the selected features.

  Type: function

bag_forest (`X`, `y`)

Forest growing algorithm that uses the class attribute max_depth as a hyperparameter. Adds trees of increasing depth to a bagged ensemble until max_depth is reached. The number of trees to add at each depth level is determined by checking if the training error converges.

Parameters:
- **X** (*pandas.dataFrame*) – Input data
- **y** (*numpy 1darray or pandas.Series*) – Input targets

bagboost_forest (X, y)

Bag-boosting forest growing algorithm, no hyperparameters needed. The number of trees to grow at each boosting iteration is determined by the convergence of the training error. Out-of-bag error is used to determine how many boosting iterations to conduct.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.

double_bagboost_forest (X, y)

Double bag-boosting forest growing algorithm, no hyperparameters needed. The number of trees to grow at each boosting iteration is determined by the convergence of the training error. Out-of-bag error is used to determine how many boosting iterations to conduct.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.

fit (X, y, `costs`=[], `groups`=[], `sketching`=1.0)

Wrapper function, builds a forest and solves LASSO optimization problem to select a subforest. Trains a final model polished model on the selected features.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets
- **costs** (*numpy 1darray*) – List of feature costs.
- **groups** (*numpy 1darray*) – List of feature groupings.
- **sketching** (*float*) – Proportion of training data to use for gaussian sketching.

fit_cv (X, y, `nfolds`=5, `n_alphas`=50, `eps`=0.001, `show_plot`=True, `kwargs`={})

Compute the entire lasso path and select the best parameter using a nfold cross validation. Returns the best regularization parameter, support size, and selected features

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.
- **nfolds** (*int*) – Number of folds for cross validation.
- **n_alphas** (*int*) – Number of alpha values to evaluate in the path.
- **eps** (*float*) – Ratio for computing the length of the path, indicates the lower bound of the log-space.
- **show_plot** (*bool*) – Displays cross validation plot with optimal number of features.
- **kwargs** (*dict*) – Additional parameters for the lasso solver.

Returns: **best_alpha** (*float*) – Best value of alpha selected by cross validation. **best_nfeats** (*int*) – Number of features selected by LASSO on the best value of alpha. **best_feats** (*numpy 1darray*) – Array of best features.

fit_transform (X, y)

Returns dataframe of selected features. :param X: Input data. :type X: pandas.dataFrame :param y: Input targets. :type y: numpy 1darray or pandas.Series

Returns: **sparse_data** – DataFrame containing selected features.

Return type: pandas.dataFrame

predict (X)

Returns binary predictions of polished model trained on selected features. :param X: Input data. :type X: pandas.dataFrame

Returns: **pred** – Array of predictions.

Return type: numpy 1darray

predict_proba (X)

Returns class probabilities of polished model trained on selected features. :param X: Input data. :type X: pandas.dataFrame

Returns: **pred** – Array of predictions.

Return type: numpy 1darray

solve_lasso (`costs=[]`, `groups=[]`, `sketching=1.0`)

Solves LASSO optimization problem using class attribute alpha as the regularization parameter. Stores the selected features, weights, and subforest.

Parameters:

- **costs** (*numpy 1darray*) – List of feature costs.
- **groups** (*numpy 1darray*) – List of feature groupings.
- **sketching** (*float*) – Proportion of training data to use for gaussian sketching.

solve_lasso_path (`X`, `y`, `n_alphas=50`, `eps=0.001`, `costs=[]`, `groups=[]`, `kwargs={}`)

Compute the entire lasso regularization path using warm start continuation. Returns a list of alphas and an numpy array of fitted coefficients.

Parameters:

- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.
- **n_alphas** (*int*) – Number of alpha values to evaluate in the path.
- **eps** (*float*) – Ratio for computing the length of the path, indicates the lower bound of the log-space.
- **costs** (*numpy 1darray*) – List of feature costs.
- **groups** (*numpy 1darray*) – List of feature groupings.
- **kwargs** (*dict*) – Additional parameters for the lasso solver.

Returns: **alphas** (*numpy 1darray*) – Array of alpha values evaluated. **coef_path** (*numpy ndarray*) – Array of LASSO solution coefficients.

class ControlBurn.ControlBurnModel.ControlBurnRegressor (`alpha=0.1`, `max_depth=10`, `build_forest_method='bagboost'`, `polish_method=RandomForestRegressor(max_features='sqrt')`, `custom_forest=[]`)

alpha

Regularization hyperparameter.

Type: float

max_depth

Maximum depth of trees to grow in the ensemble.

Type: float

threshold

Convergence threshold.

Type: float

tail

Number of iterations to check convergence on.

Type: int

forest

List of DecisionTreeRegressor objects. Represents the full ensemble.

Type: list

subforest
List of DecisionTreeRegressor objects. Represents the sparse ensemble.

Type: list

features_selected_
List of features selected by the subforest.

Type: list

feature_importances_
List of feature importance scores computed from the subforest.

Type: list

weights
Solution coefficients to the LASSO optimization problem.

Type: list

alpha_range
Range of alphas enumerated when computing the regularization path.

coef_path
LASSO solution coefficients obtained by computing the regularization path.

polish_method
Method used to polish the final model on the selected features.

Type: function

bag_forest (`X`, `y`)
Forest growing algorithm that uses the class attribute max_depth as a hyperparameter. Adds trees of increasing depth to a bagged ensemble until max_depth is reached. The number of trees to add at each depth level is determined by checking if the training error converges.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.

bagboost_forest (`X`, `y`)
Bag-boosting forest growing algorithm, no hyperparameters needed. The number of trees to grow at each boosting iteration is determined by the convergence of the training error. Out-of-bag error is used to determine how many boosting iterations to conduct.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.

double_bagboost_forest (`X`, `y`)
Double bag-boosting forest growing algorithm, no hyperparameters needed. The number of trees to grow at each boosting iteration is determined by the convergence of the training error. Out-of-bag error is used to determine how many boosting iterations to conduct.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.

fit (`X`, `y`, `costs=[]`, `groups=[]`, `sketching=1.0`)
Wrapper function, builds a forest and solves LASSO optimization problem to select a subforest. Trains a final model polished model on the selected features.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets
- **costs** (*numpy 1darray*) – List of feature costs.
- **groups** (*numpy 1darray*) – List of feature groupings.
- **sketching** (*float*) – Proportion of training data to use for gaussian sketching.

fit_cv (X, y, nfolds=5, n_alphas=500, show_plot=True, kwargs={})

Compute the entire lasso path and select the best parameter using a nfold cross validation. Returns the best regularization parameter, support size, and selected features

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.
- **nfolds** (*int*) – Number of folds for cross validation.
- **n_alphas** (*int*) – Number of alpha values to evaluate in the path.
- **show_plot** (*bool*) – Displays cross validation plot with optimal number of features.
- **kwargs** (*dict*) – Additional parameters for the lasso solver.

Returns: **best_alpha** (*float*) – Best value of alpha selected by cross validation. **best_nfeats** (*int*) – Number of features selected by LASSO on the best value of alpha. **best_feats** (*numpy 1darray*) – Array of best features.

fit_transform (X, y)

Returns dataframe of selected features.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.

Returns: **sparse_data** – DataFrame containing selected features.

Return type: pandas.dataFrame

predict (X)

Returns predictions of polished model trained on selected features.

Parameters: **X** (*pandas.dataFrame*) – Input data.

Returns: **pred** – Array of predictions.

Return type: numpy 1darray

solve_l0 (X, y, K, verbose=False)

Selects the best subset of trees in the ensemble such that the total features used is equal to K. Best subset solver implemented using gurobi. Dependencies imported inside package since optional function.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.
- **K** (*int*) – Number of features to select.
- **verbose** (*bool*) – Prints Gurobi output.

solve_lasso (costs=[], groups=[], sketching=1.0)

Solves LASSO optimization problem using class attribute alpha as the regularization parameter. Stores the selected features, weights, and subforest.

Parameters:
- **costs** (*numpy 1darray*) – List of feature costs.
- **groups** (*numpy 1darray*) – List of feature groupings.
- **sketching** (*float*) – Proportion of training data to use for gaussian sketching.

solve_lasso_path (X, y, n_alphas=500, costs=[], groups=[], kwargs={})
Compute the entire lasso regularization path using warm start continuation. Returns a list of alphas and an numpy array of fitted coefficients.

Parameters:
- **X** (*pandas.dataFrame*) – Input data.
- **y** (*numpy 1darray or pandas.Series*) – Input targets.
- **n_alphas** (*int*) – Number of alpha values to evaluate in the path.
- **costs** (*numpy 1darray*) – List of feature costs.
- **groups** (*numpy 1darray*) – List of feature groupings.
- **kwargs** (*dict*) – Additional parameters for the lasso solver.

Returns: **alphas** (*numpy 1darray*) – Array of alpha values evaluated. **coef_path** (*numpy ndarray*) – Array of LASSO solution coefficients.

ControlBurnInterpret.py

This module contains functions to interpret fitted ControlBurn objects.

class ControlBurn.ControlBurnInterpret.InterpretClassifier (ControlBurnClassifier, X, y)

cb
Fitted ControlBurnClassifier object.

Type: ControlBurnClassifier

X
Input data.

Type: pandas.dataFrame

y
Input targets.

Type: numpy 1darray or pandas Series:

list_subforest (show_plot=False)
Lists the features used in each tree of the selected subforest to give a sense of model structure, returns the array of features used.

Parameters: **show_plot** (*bool*) – Displays visualization.

Returns: **features_used** – List of features used each tree in the selected model.

Return type: list

plot_feature_importances (groups=[], group_names=[], show_plot=False)
Plots a bar plot of the weighted feature importance scores in the subforest.

Parameters:
- **show_plot** (*bool*) – Displays visualization.
- **groups** (*list*) – List that indicates which features belong to which groups
- **group_names** (*list*) – List of group names

Returns: **feature_importances** – List of features in the subforest with associated feature importance scores

Return type: numpy.2darray

plot_pairwise_interactions (feature1, feature2, show_plot=True)
Plot a heatmap showing impact of pairwise interaction on the response.

Parameters:
- **feature1** (*string*) – Feature name.
- **feature2** (*string*) – Feature name.
- **show_plot** (*bool*) – Displays visualization.

Returns: **pairwise_df** – Dataframe of pairwise interaction contributions.

Return type: pandas.DataFrame

plot_regularization_path (`show_plot`=True, `more_colors`=False)

Plot the LASSO regularization path for a ControlBurnRegressor. Feature importance for a feature computed as the weighted sum of feature importances for each tree in the selected subforest.

Parameters:

- **show_plot** (*bool*) – Displays visualization.
- **more_colors** (*int*) – Number of additional colors to include.

Returns: **regularization_df** – Dataframe of contributions along regularization path.

Return type: pandas.DataFrame

plot_single_feature_shape (`feature`, `show_plot`=True)

Plot a shape function that demonstrates the contribution of single feature trees on the response.

Parameters:

- **feature** (*string*) – Feature name.
- **show_plot** (*bool*) – Displays visualization.

class ControlBurn.ControlBurnInterpret.InterpretRegressor (`ControlBurnRegressor`, `X`, `y`)

cb

Fitted ControlBurnRegressor object.

Type: ControlBurnRegressor

X

Input data.

Type: pandas.dataFrame

y

Input targets.

Type: numpy 1darray or pandas Series:

list_subforest (`show_plot`=False)

Lists the features used in each tree of the selected subforest to give a sense of model structure, returns the array of features used.

Parameters: **show_plot** (*bool*) – Displays visualization.

Returns: **features_used** – List of features used each tree in the selected model.

Return type: list

plot_feature_importances (`groups`=[], `group_names`=[], `show_plot`=False)

Plots a bar plot of the weighted feature importance scores in the subforest.

Parameters:

- **show_plot** (*bool*) – Displays visualization.
- **groups** (*list*) – List that indicates which features belong to which groups
- **group_names** (*list*) – List of group names

Returns: **feature_importances** – List of features in the subforest with associated feature importance scores

Return type: numpy.2darray

plot_pairwise_interactions (`feature1`, `feature2`, `show_plot`=True)

Plot a heatmap showing impact of pairwise interaction on the response.

Parameters:

- **feature1** (*string*) – Feature name.
- **feature2** (*string*) – Feature name.
- **show_plot** (*bool*) – Displays visualization.

Returns: **pairwise_df** – Dataframe of pairwise interaction contributions.

Return type: pandas.DataFrame

plot_regularization_path (`show_plot`=True, `more_colors`=False)

Plot the LASSO regularization path for a ControlBurnRegressor. Feature importance for a feature computed as the weighted sum of feature importances for each tree in the selected subforest.

Parameters:
- **show_plot** (*bool*) – Displays visualization.
- **more_colors** (*int*) – Number of additional colors to include.

Returns: **regularization_df** – Dataframe of contributions along regularization path.

Return type: pandas.DataFrame

plot_single_feature_shape (`feature`, `show_plot`=True)

Plot a shape function that demonstrates the contribution of single feature trees on the response.

Parameters:
- **feature** (*string*) – Feature name.
- **show_plot** (*bool*) – Displays visualization.

Indices and tables

- genindex
- modindex
- search

Index

Python Module Index

c