

# RegressionCV

**mod** regression\_cv

Definition for RegressionCV.

**class** RegressionCV

Bases: BaseAutoCV, RegressorMixin, ExplainerMixin

Defines an auto regression tree, based on the bayesian optimization base class.

Source code in `src/tree_machine/regression_cv.py`



91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141

▼ Details

14214314414514614714814915015115215315415515615715815916016116216316416516616716816917017117217317417517

**attr** **scorer** property

```
scorer
```

Returns correct scorer to use when scoring with RegressionCV.

**meth** **\_\_init\_\_**

```
__init__(metric, cv, n_trials, timeout, config)
```

Constructor for RegressionCV.

**Parameters:**

Name	Type	Description	Default
<code>metric</code>	<code>AcceptableRegression</code>	Loss metric to use as base for estimation process.	<i>required</i>
<code>cv</code>	<code>BaseCrossValidator</code>	Splitter object to use when estimating the model.	<i>required</i>
<code>n_trials</code>	<code>NonNegativeInt</code>	Number of optimization trials to use when finding a model.	<i>required</i>
<code>timeout</code>	<code>NonNegativeInt</code>	Timeout in seconds to stop the optimization.	<i>required</i>
<code>config</code>	<code>RegressionCVConfig</code>	Configuration to use when fitting the model.	<i>required</i>

Source code in `src/tree_machine/regression_cv.py`

```
100 @validate_call(config={"arbitrary_types_allowed": True})
101 def __init__(
102     self,
103     metric: AcceptableRegression,
104     cv: BaseCrossValidator,
105     n_trials: NonNegativeInt,
106     timeout: NonNegativeInt,
107     config: RegressionCVConfig,
108 ) -> None:
109     """
110     Constructor for RegressionCV.
111
112     Args:
113         metric: Loss metric to use as base for estimation process.
114         cv: Splitter object to use when estimating the model.
115         n_trials: Number of optimization trials to use when finding a model.
116         timeout: Timeout in seconds to stop the optimization.
117         config: Configuration to use when fitting the model.
118     """
119     super().__init__(metric, cv, n_trials, timeout)
120     self.config = config
```

**meth** `explain`

```
explain(X, **explainer_params)
```

Explains the inputs.

” Source code in `src/tree_machine/regression_cv.py`

```
122 def explain(self, X: Inputs, **explainer_params) -> dict[str,  
123         NumpyArray[np.float64]]:  
124     """  
125     Explains the inputs.  
126     """  
127  
128     self.model_  
129     check_is_fitted(self, "model_", msg="Model is not fitted.")  
130  
131     if getattr(self, "explainer_", None) is None:  
132         self.explainer_ = TreeExplainer(self.model_, **explainer_params)  
133  
134     return {  
135         "mean_value": self.explainer_.expected_value,  
136         "shap_values": self.explainer_.shap_values(self._validate_X(X)),  
    }
```

#### meth fit

```
fit(X, y, **fit_params)
```

Fits RegressionCV.

#### Parameters:

Name	Type	Description	Default
<code>X</code>	<code>Inputs</code>	input data to use in fitting trees.	<i>required</i>
<code>y</code>	<code>GroundTruth</code>	actual targets for fitting.	<i>required</i>

Source code in `src/tree_machine/regression_cv.py`

```
138 def fit(self, X: Inputs, y: GroundTruth, **fit_params) -> "RegressionCV":
139     """
140     Fits RegressionCV.
141
142     Args:
143         X: input data to use in fitting trees.
144         y: actual targets for fitting.
145     """
146     self.feature_names_ = list(X.columns) if isinstance(X, pd.DataFrame) else
147     []
148     constraints = self.config.get_kwargs(self.feature_names_)
149
150     self.model_ = self.optimize(
151         estimator_type=XGBRegressor,
152         X=self._validate_X(X),
153         y=self._validate_y(y),
154         parameters=self.config.parameters,
155         return_train_score=self.config.return_train_score,
156         **constraints,
157     )
158     self.feature_importances_ = self.model_.feature_importances_
159
160     return self
```

meth `predict`

`predict(X)`

Returns model predictions.

Source code in `src/tree_machine/regression_cv.py`

```
161 def predict(self, X: Inputs) -> Predictions:
162     """
163     Returns model predictions.
164     """
165     check_is_fitted(self, "model_", msg="Model is not fitted.")
166     return self.model_.predict(self._validate_X(X))
```

meth `predict_proba`

`predict_proba(X)`

Returns model probability predictions.

” Source code in `src/tree_machine/regression_cv.py`

```
168 def predict_proba(self, X: Inputs) -> Predictions:
169     """
170     Returns model probability predictions.
171     """
172     raise NotImplementedError("Not implemented for RegressionCV.")
```

### **class** RegressionCVConfig

Available config to use when fitting a regression model.

 **dictionary containing monotonicity direction allowed for each**

variable. 0 means no monotonicity, 1 means increasing and -1 means decreasing monotonicity.

interactions: list of lists containing permitted relationships in data. parameters: dictionary with distribution bounds for each hyperparameter to search on during optimization. n\_jobs: Number of jobs to use when fitting the model.



Source code in `src/tree_machine/regression_cv.py`

```
33 @dataclass(frozen=True, config={"arbitrary_types_allowed": True})
34 class RegressionCVConfig:
35     """
36     Available config to use when fitting a regression model.
37
38     monotone_constraints: dictionary containing monotonicity direction allowed
39     for each
40     variable. 0 means no monotonicity, 1 means increasing and -1 means
41     decreasing
42     monotonicity.
43     interactions: list of lists containing permitted relationships in data.
44     parameters: dictionary with distribution bounds for each hyperparameter to
45     search
46     on during optimization.
47     n_jobs: Number of jobs to use when fitting the model.
48     """
49
50     monotone_constraints: dict[str, int]
51     interactions: list[list[str]]
52     n_jobs: int
53     parameters: OptimizerParams
54     return_train_score: bool
55
56     def get_kwargs(self, feature_names: list[str]) -> dict:
57         """
58         Returns parsed and validated constraint configuration for a
59         RegressionCV model.
60
61         Args:
62             feature_names: list of feature names. If empty, will return empty
63             constraints dictionaries and lists.
64         """
65         return {
66             "monotone_constraints": {
67                 feature_names.index(key): value
68                 for key, value in self.monotone_constraints.items()
69             },
70             "interaction_constraints": [
71                 [feature_names.index(key) for key in lt] for lt in
72                 self.interactions
73             ],
74             "n_jobs": self.n_jobs,
75         }
```

meth `get_kwargs`

```
get_kwargs(feature_names)
```

Returns parsed and validated constraint configuration for a RegressionCV model.

**Parameters:**

Name	Type	Description	Default
<code>feature_names</code>	<code>list[str]</code>	list of feature names. If empty, will return empty constraints dictionaries and lists.	<i>required</i>

” Source code in `src/tree_machine/regression_cv.py`

```
53 def get_kwargs(self, feature_names: list[str]) -> dict:
54     """
55     Returns parsed and validated constraint configuration for a RegressionCV
56     model.
57
58     Args:
59         feature_names: list of feature names. If empty, will return empty
60         constraints dictionaries and lists.
61     """
62     return {
63         "monotone_constraints": {
64             feature_names.index(key): value
65             for key, value in self.monotone_constraints.items()
66         },
67         "interaction_constraints": [
68             [feature_names.index(key) for key in lt] for lt in
69 self.interactions
70         ],
71         "n_jobs": self.n_jobs,
72     }
```