

ClassifierCV

mod classifier_cv

Definition for ClassifierCV.

class ClassifierCV

Bases: [BaseAutoCV](#), [ClassifierMixin](#), [ExplainerMixIn](#)

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

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



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
142
143

▼ Details

14414514614714814915015115215315415515615715815916016116216316416516616716816917017117217317417517617717

attr **scorer** property

```
scorer
```

Returns correct scorer to use when scoring with RegressionCV.

meth **__init__**

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

Constructor for ClassifierCV.

Parameters:

Name	Type	Description	Default
<code>metric</code>	<code>AcceptableClassifier</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>ClassifierCVConfig</code>	Configuration to use when fitting the model.	<i>required</i>

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

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

meth `explain`

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

Explains the inputs.

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

```
124 def explain(self, X: Inputs, **explainer_params) -> dict[str,  
125         NumpyArray[np.float64]]:  
126     """  
127     Explains the inputs.  
128     """  
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     shap_values = self.explainer_.shap_values(self._validate_X(X))  
135     shape = shap_values.shape  
136  
137     return {  
138         "mean_value": self.explainer_.expected_value,  
139         "shap_values": shap_values.reshape(shape[0], shape[1], -1),  
    }
```

meth `fit`

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

Fits ClassifierCV.

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/classifier_cv.py`

```
141 def fit(self, X: Inputs, y: GroundTruth, **fit_params) -> "ClassifierCV":
142     """
143     Fits ClassifierCV.
144
145     Args:
146         X: input data to use in fitting trees.
147         y: actual targets for fitting.
148     """
149     self.feature_names_ = list(X.columns) if isinstance(X, pd.DataFrame) else
150     []
151     constraints = self.config.get_kwargs(self.feature_names_)
152
153     self.model_ = self.optimize(
154         estimator_type=XGBClassifier,
155         X=self._validate_X(X),
156         y=self._validate_y(y),
157         parameters=self.config.parameters,
158         return_train_score=self.config.return_train_score,
159         **constraints,
160     )
161     self.feature_importances_ = self.model_.feature_importances_
162
163     return self
```

meth `predict`

```
predict(X)
```

Returns model predictions.

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

```
164 def predict(self, X: Inputs) -> Predictions:
165     """
166     Returns model predictions.
167     """
168     check_is_fitted(self, "model_", msg="Model is not fitted.")
169     return self.model_.predict(self._validate_X(X))
```

meth `predict_proba`

```
predict_proba(X)
```

Returns model probability predictions.

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

```
171 def predict_proba(self, X: Inputs) -> Predictions:
172     """
173     Returns model probability predictions.
174     """
175     check_is_fitted(self, "model_", msg="Model is not fitted.")
176     return self.model_.predict_proba(self._validate_X(X))
```

`class` ClassifierCVConfig

Available config to use when fitting a classification 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. n_jobs: Number of jobs to use when fitting the model. parameters: dictionary with distribution bounds for each hyperparameter to search on during optimization. return_train_score: whether to return the train score when fitting the model.

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

```
35 @dataclass(frozen=True, config={"arbitrary_types_allowed": True})
36 class ClassifierCVConfig:
37     """
38     Available config to use when fitting a classification model.
39
40     monotone_constraints: dictionary containing monotonicity direction allowed
41     for each
42     variable. 0 means no monotonicity, 1 means increasing and -1 means
43     decreasing
44     monotonicity.
45     interactions: list of lists containing permitted relationships in data.
46     n_jobs: Number of jobs to use when fitting the model.
47     parameters: dictionary with distribution bounds for each hyperparameter to
48     search
49     on during optimization.
50     return_train_score: whether to return the train score when fitting the
51     model.
52     """
53
54     monotone_constraints: dict[str, int]
55     interactions: list[list[str]]
56     n_jobs: int
57     parameters: OptimizerParams
58     return_train_score: bool
59
60     def get_kwargs(self, feature_names: list[str]) -> dict:
61         """
62         Returns parsed and validated constraint configuration for a
63         ClassifierCV model.
64
65         Args:
66             feature_names: list of feature names. If empty, will return empty
67             constraints dictionaries and lists.
68         """
69         return {
70             "monotone_constraints": {
71                 feature_names.index(key): value
72                 for key, value in self.monotone_constraints.items()
73             },
74             "interaction_constraints": [
75                 [feature_names.index(key) for key in lt] for lt in
76                 self.interactions
77             ],
78             "n_jobs": self.n_jobs,
79         }
```

meth `get_kwargs`

```
get_kwargs(feature_names)
```

Returns parsed and validated constraint configuration for a ClassifierCV 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/classifier_cv.py`

```
56 def get_kwargs(self, feature_names: list[str]) -> dict:
57     """
58     Returns parsed and validated constraint configuration for a ClassifierCV
59     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     }
```