

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`



85
86
87
88
89
90
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

▼ Details

13613713813914014114214314414514614714814915015115215315415515615715815916016116216316416516616716816917

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`

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

meth `explain`

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

Explains the inputs.

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

```
116 def explain(self, X: Inputs, **explainer_params) -> dict[str,  
117         NumpyArray[np.float64]]:  
118     """  
119     Explains the inputs.  
120     """  
121     check_is_fitted(self, "model_", msg="Model is not fitted.")  
122  
123     if getattr(self, "explainer_", None) is None:  
124         self.explainer_ = TreeExplainer(self.model_, **explainer_params)  
125  
126     shap_values = self.explainer_.shap_values(self._validate_X(X))  
127     shape = shap_values.shape  
128  
129     return {  
130         "mean_value": self.explainer_.expected_value,  
131         "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`

```
133 def fit(self, X: Inputs, y: GroundTruth, **fit_params) -> "ClassifierCV":
134     """
135     Fits ClassifierCV.
136
137     Args:
138         X: input data to use in fitting trees.
139         y: actual targets for fitting.
140     """
141     self.feature_names_ = list(X.columns) if isinstance(X, pd.DataFrame) else
142     []
143     constraints = self.config.get_kwargs(self.feature_names_)
144
145     self.model_ = self.optimize(
146         estimator_type=XGBClassifier,
147         X=self._validate_X(X),
148         y=self._validate_y(y),
149         parameters=self.config.parameters,
150         return_train_score=self.config.return_train_score,
151         **constraints,
152     )
153     self.feature_importances_ = self.model_.feature_importances_
154
155     return self
```

meth `predict`

```
predict(X)
```

Returns model predictions.

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

```
156 def predict(self, X: Inputs) -> Predictions:
157     """
158     Returns model predictions.
159     """
160     check_is_fitted(self, "model_", msg="Model is not fitted.")
161     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`

```
163 def predict_proba(self, X: Inputs) -> Predictions:
164     """
165     Returns model probability predictions.
166     """
167     check_is_fitted(self, "model_", msg="Model is not fitted.")
168     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. parameters: dictionary with distribution bounds for each hyperparameter to search on during optimization. n_jobs: Number of jobs to use when fitting the model. sampler: `imblearn` sampler to use when fitting models.

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

```
27 @dataclass(frozen=True, config={"arbitrary_types_allowed": True})
28 class ClassifierCVConfig:
29     """
30     Available config to use when fitting a classification model.
31
32     monotone_constraints: dictionary containing monotonicity direction allowed
33     for each
34         variable. 0 means no monotonicity, 1 means increasing and -1 means
35     decreasing
36         monotonicity.
37     interactions: list of lists containing permitted relationships in data.
38     parameters: dictionary with distribution bounds for each hyperparameter to
39     search
40         on during optimization.
41     n_jobs: Number of jobs to use when fitting the model.
42     sampler: `imblearn` sampler to use when fitting models.
43     """
44
45     monotone_constraints: dict[str, int]
46     interactions: list[list[str]]
47     n_jobs: int
48     parameters: OptimizerParams
49     return_train_score: bool
50
51     def get_kwargs(self, feature_names: list[str]) -> dict:
52         """
53         Returns parsed and validated constraint configuration for a
54         ClassifierCV model.
55
56         Args:
57             feature_names: list of feature names. If empty, will return empty
58                 constraints dictionaries and lists.
59         """
60         return {
61             "monotone_constraints": {
62                 feature_names.index(key): value
63                 for key, value in self.monotone_constraints.items()
64             },
65             "interaction_constraints": [
66                 [feature_names.index(key) for key in lt] for lt in
67                 self.interactions
68             ],
69             "n_jobs": self.n_jobs,
70         }
```

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`

```
48 def get_kwargs(self, feature_names: list[str]) -> dict:
49     """
50     Returns parsed and validated constraint configuration for a ClassifierCV
51     model.
52
53     Args:
54         feature_names: list of feature names. If empty, will return empty
55         constraints dictionaries and lists.
56     """
57     return {
58         "monotone_constraints": {
59             feature_names.index(key): value
60             for key, value in self.monotone_constraints.items()
61         },
62         "interaction_constraints": [
63             [feature_names.index(key) for key in lt] for lt in
64 self.interactions
65         ],
66         "n_jobs": self.n_jobs,
67     }
```