

Base

mod base

Definition of BaseAutoCV.

class BaseAutoCV

Bases: `ABC`, `BaseEstimator`

Defines BaseAutoCV, a class to help fit models using Bayesian optimization.

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



23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73

▼ Details

99100101102103104105106107108109110111112113114115116117118119120121122123124125126127128129130131132133

attr **cv_results** property

```
cv_results
```

Returns test score for each fold for the model best estimator.

attr **scorer** abstractmethod property

```
scorer
```

Abstract implementation for a function that returns the correct scorer to use when fitting/scoring models.

meth **__init__**

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

Constructor for BaseAutoTreeCV.

Parameters:

Name	Type	Description	Default
<code>metric</code>	<code>str</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>

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

```
39 | @validate_call(config={"arbitrary_types_allowed": True})
40 | def __init__(
41 |     self,
42 |     metric: str,
43 |     cv: BaseCrossValidator,
44 |     n_trials: NonNegativeInt,
45 |     timeout: NonNegativeInt,
46 | ) -> None:
47 |     """
48 |     Constructor for BaseAutoTreeCV.
49 |
50 |     Args:
51 |         metric: Loss metric to use as base for estimation process.
52 |         cv: Splitter object to use when estimating the model.
53 |         n_trials: Number of optimization trials to use when finding a model.
54 |         timeout: Timeout in seconds to stop the optimization.
55 |     """
56 |     self.metric = metric
57 |     self.cv = cv
58 |     self.n_trials = n_trials
59 |     self.timeout = timeout
```

meth `explain`

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

Explains the inputs.

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

```
61 | def explain(self, X: Inputs, **explainer_params):
62 |     """
63 |     Explains the inputs.
64 |     """
65 |     raise NotImplementedError()
```

meth `optimize`

```
optimize(estimator_type, X, y, parameters, return_train_score, **kwargs)
```

Fits a model using Bayesian optimization and optuna.

Parameters:

Name	Type	Description	Default
<code>estimator_type</code>		type of object to use when fitting models.	<i>required</i>
<code>X</code>	<code>Inputs</code>	Input data to use when fitting models.	<i>required</i>
<code>y</code>	<code>GroundTruth</code>	Ground truth data to use when fitting models.	<i>required</i>
<code>return_train_score</code>	<code>bool</code>	Whether to return or not the training score for optimization.	<i>required</i>
<code>parameters</code>	<code>OptimizerParams</code>	Distributions defined by user to select trial values.	<i>required</i>

Returns:

Type	Description
	Fitted <code>estimator_type</code> object, using the best parameters selected using Bayesian optimization.

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



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
144
145
146
147
148
149
150
151
152

▼ Details

153154

▼ Details

```
def optimize(
    self,
    estimator_type,
    X: Inputs,
    y: GroundTruth,
    parameters: OptimizerParams,
    return_train_score: bool,
    **kwargs,
):
    """
    Fits a model using Bayesian optimization and optuna.

    Args:
        estimator_type: type of object to use when fitting models.
        X: Input data to use when fitting models.
        y: Ground truth data to use when fitting models.
        return_train_score: Whether to return or not the training score for
            optimization.
        parameters: Distributions defined by user to select trial values.

    Returns:
        Fitted `estimator_type` object, using the best parameters selected using
        Bayesian optimization.
    """

    def _objective(trial: Trial) -> float:
        """Objective function to use in optimization."""
        estimator = estimator_type(
            **kwargs,
            **parameters.get_trial_values(trial),
        )

        cv_results = cross_validate(
            estimator,
            X,
            y,
            scoring=self.scorer,
            cv=self.cv,
            return_train_score=return_train_score,
        )
        trial.set_user_attr("cv_results", cv_results)
        return np.mean(cv_results["test_score"])

    self.study_ = create_study(
        direction="maximize",
        sampler=TPESampler(),
        pruner=HyperbandPruner(),
    )
    self.study_.optimize(_objective, n_trials=self.n_trials,
        timeout=self.timeout)
    self.best_params_ = self.study_.best_params

    return estimator_type(**self.best_params_, **kwargs).fit(X, y)
```

meth **predict** abstractmethod

```
predict(X)
```

Abstract implementation for a prediction function.

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

```
67 | @abstractmethod
68 | def predict(self, X: Inputs) -> Predictions:
69 |     """
70 |     Abstract implementation for a prediction function.
71 |     """
72 |     raise NotImplementedError()
```

meth **predict_proba** abstractmethod

```
predict_proba(X)
```

Abstract implementation for a prediction function, returning probabilities.

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

```
74 | @abstractmethod
75 | def predict_proba(self, X: Inputs) -> Predictions:
76 |     """
77 |     Abstract implementation for a prediction function, returning
78 |     probabilities.
79 |     """
    |     raise NotImplementedError()
```