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`

▼ Details


```
991001011021031041051061071081091101111121131141151161171181191201211221231241251261271281291301311321331
```

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
`metric`	`str`	Loss metric to use as base for estimation process.	*required*
`cv`	`BaseCrossValidator`	Splitter object to use when estimating the model.	*required*
`n_trials`	`NonNegativeInt`	Number of optimization trials to use when finding a model.	*required*
`timeout`	`NonNegativeInt`	Timeout in seconds to stop the optimization.	*required*

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

```python
@validate_call(config={"arbitrary_types_allowed": True})
def __init__(
    self,
    metric: str,
    cv: BaseCrossValidator,
    n_trials: NonNegativeInt,
    timeout: NonNegativeInt,
) -> None:
    """
    Constructor for BaseAutoTreeCV.

    Args:
        metric: Loss metric to use as base for estimation process.
        cv: Splitter object to use when estimating the model.
        n_trials: Number of optimization trials to use when finding a model.
        timeout: Timeout in seconds to stop the optimization.
    """
    self.metric = metric
    self.cv = cv
    self.n_trials = n_trials
    self.timeout = timeout
```

`meth` explain

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

Explains the inputs.

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

```python
def explain(self, X: Inputs, **explainer_params):
    """
    Explains the inputs.
    """
    raise NotImplementedError()
```

`meth` optimize

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

Fits a model using Bayesian optimization and optuna.

Parameters:

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

Returns:

Type	Description
	Fitted `estimator_type` object, using the best parameters selected using Bayesian optimization.

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

```
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
144
145
146
147
148
149
150
151
```

▼ Details

```
152153
```

▼ 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

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

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