QuantileCV

Auto-tuned quantile regression. Uses `RegressionCVConfig`. For examples, see the API Guide.

`mod` quantile_cv

Definition for RegressionCV.

`class` QuantileCV

Bases: `RegressionCV`

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

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

```python
class QuantileCV(RegressionCV):
    """
    Defines an auto quantile tree, based on the bayesian optimization base class.
    """

    @validate_call(config={"arbitrary_types_allowed": True})
    def __init__(
        self,
        alpha: NonNegativeFloat,
        cv: BaseCrossValidator,
        n_trials: NonNegativeInt,
        timeout: NonNegativeInt,
        config: RegressionCVConfig,
        backend: str = "xgboost",
    ) -> None:
        """
        Constructor for QuantileCV.

        Args:
            alpha: The quantile to estimate, which must be between 0 and 1.
            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.
            config: Configuration to use when fitting the model.
            backend: Backend to use for the model. Either "xgboost" or "catboost".
        """
        super().__init__("quantile", cv, n_trials, timeout, config,
backend=backend)
        self.alpha_ = alpha

    @property
    def scorer(self) -> tp.Callable[..., float]:
        """
        Returns correct scorer to use when scoring with QuantileCV.
        """
        # For quantile regression, we always use the quantile metric with alpha
parameter
        return make_scorer(
            update_wrapper(
                partial(
                    regression_metrics["quantile"],
                    alpha=self.alpha_,
                ),
                regression_metrics["quantile"],
            ),
            greater_is_better=False,
        )
```

`attr` **scorer** `property`

```
scorer
```

Returns correct scorer to use when scoring with QuantileCV.

`meth` __init__

```
__init__(alpha, cv, n_trials, timeout, config, backend='xgboost')
```

Constructor for QuantileCV.

Parameters:

Name	Type	Description	Default
`alpha`	`NonNegativeFloat`	The quantile to estimate, which must be between 0 and 1.	*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*
`config`	`RegressionCVConfig`	Configuration to use when fitting the model.	*required*
`backend`	`str`	Backend to use for the model. Either "xgboost" or "catboost".	`'xgboost'`

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

```python
@validate_call(config={"arbitrary_types_allowed": True})
def __init__(
    self,
    alpha: NonNegativeFloat,
    cv: BaseCrossValidator,
    n_trials: NonNegativeInt,
    timeout: NonNegativeInt,
    config: RegressionCVConfig,
    backend: str = "xgboost",
) -> None:
    """
    Constructor for QuantileCV.

    Args:
        alpha: The quantile to estimate, which must be between 0 and 1.
        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.
        config: Configuration to use when fitting the model.
        backend: Backend to use for the model. Either "xgboost" or "catboost".
    """
    super().__init__("quantile", cv, n_trials, timeout, config, backend=backend)
    self.alpha_ = alpha
```