SKLEARN MINI

A lightweight implementation of ML techniques

College of Computing

UM6P University Mohammed VI Polytechnic

PROJECT OVERVIEW

In this project, our main objective was to gain a deeper understanding of the internal workings of commonly used machine learning tools, rather than relying on them as black boxes.

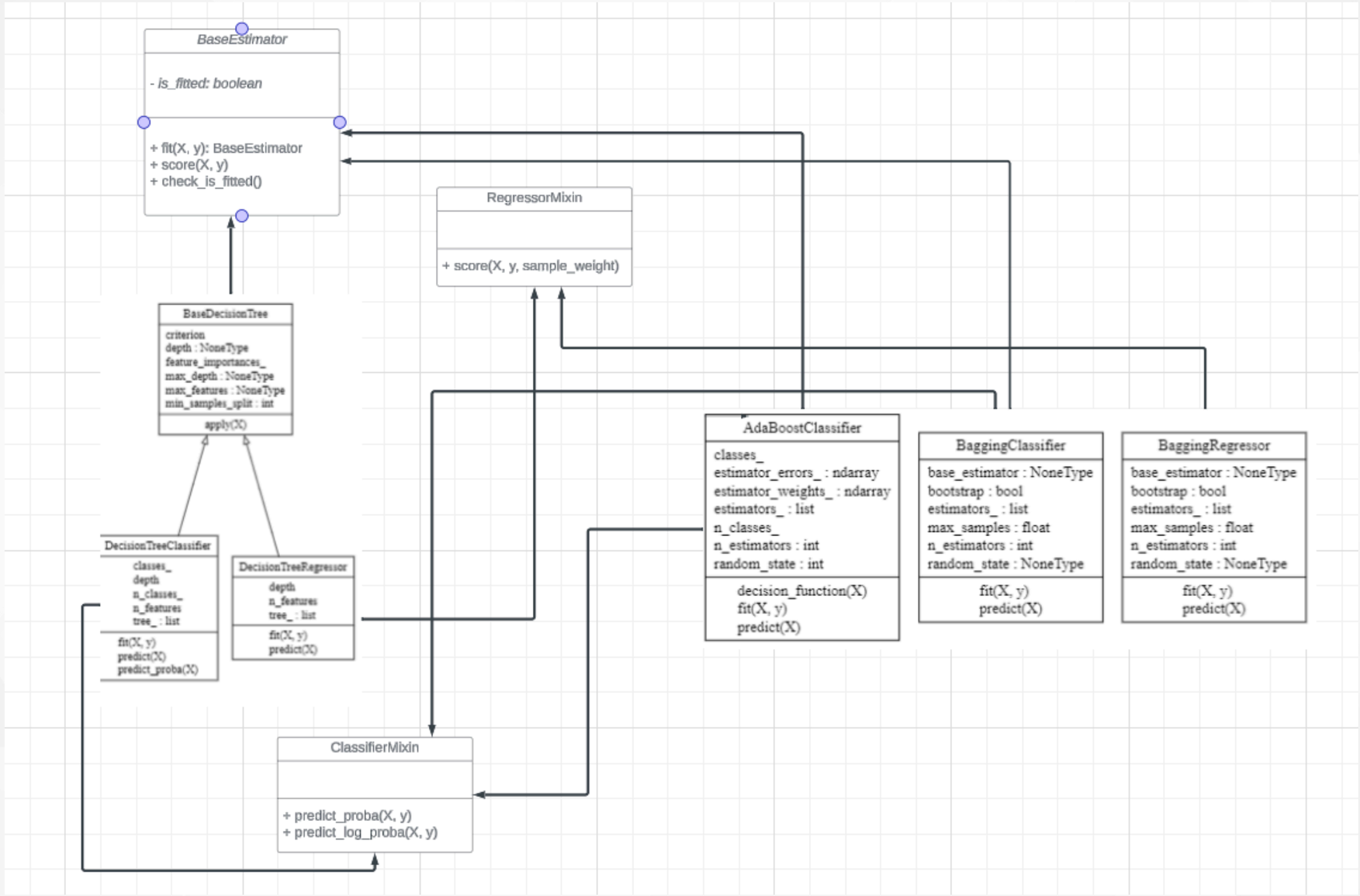
OUTLINE

- Project design(Class Diagram, Library architecture)
- Library exploration :
  - Preprocessing :
    - Feature Scaling
    - Normalization
    - Imputation
    - Encoding
    - Feature Selection
  - Supervised Learning :
    - Linear Models
    - Neighbors
    - Naive Bayes
    - Decision Trees
  - Ensemble Methods :
    - Random Forest
    - Bagging
    - Boosting
    - Voting
    - Stacking
  - Neural Networks :
    - MLP
  - Model selection
  - Metrics and Evaluaton
  - Other utilities

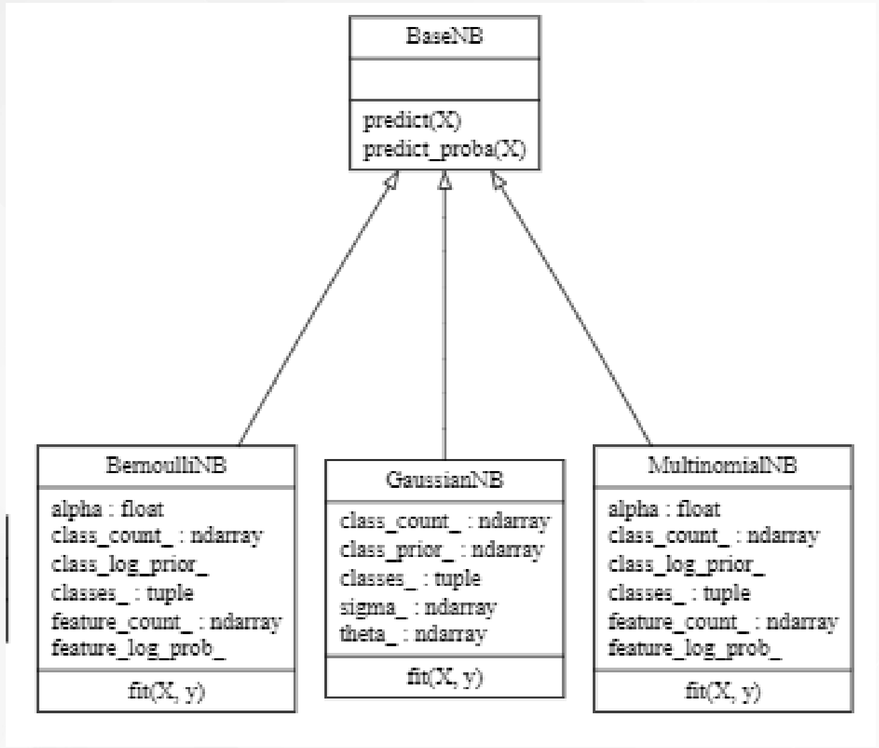
College of Computing

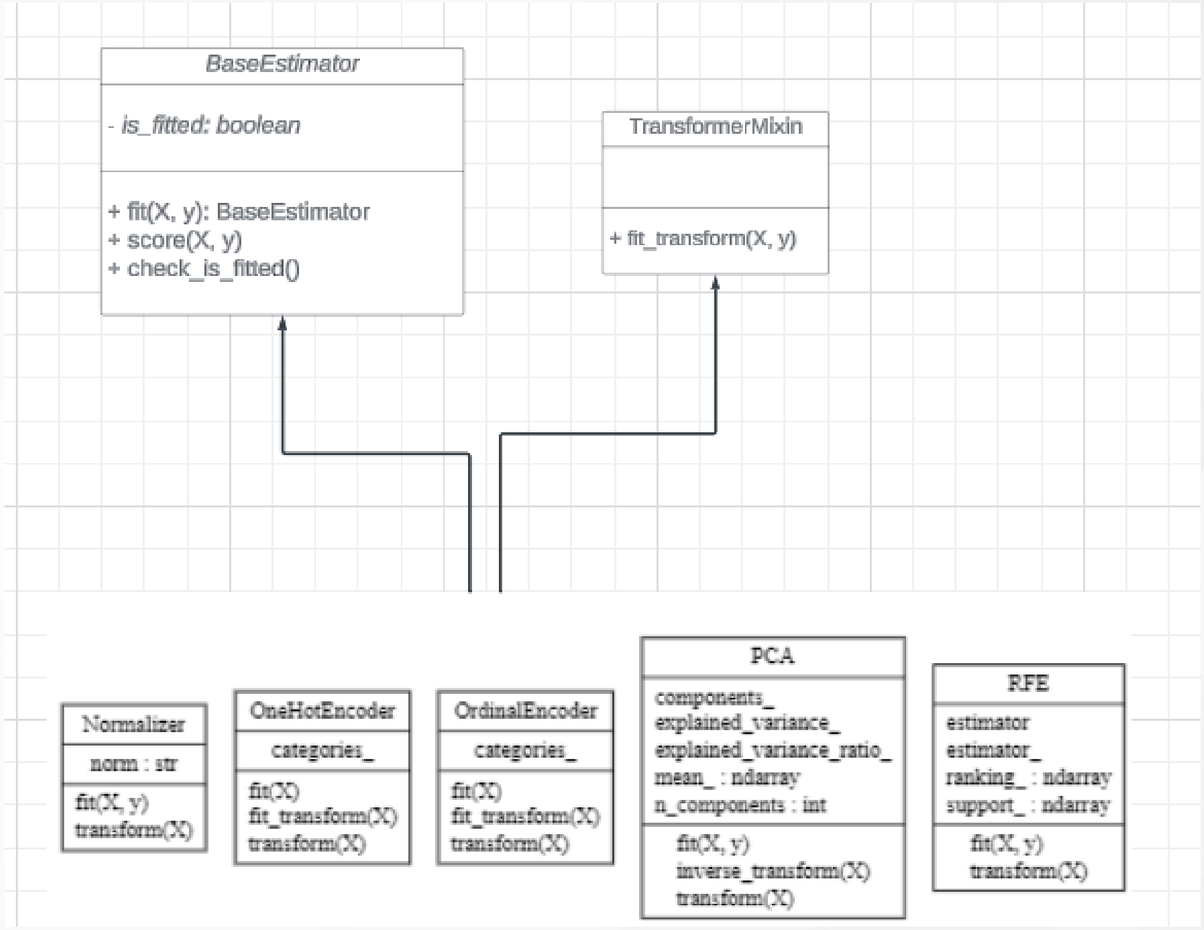
UM6P University Mohammed VI Polytechnic

PROJECT DESIGN



PROJECT DESIGN





LIBRARY ARCHITECTURE

Folder hiearchy

BASE CLASSES

```python
class ClassifierMixin:      You, 3 weeks ago • first architecture
    """Mixin class for all classifiers in mini_sklearn."""

    def predict_proba(self, X):
        """Predict class probabilities."""
        raise NotImplementedError("predict_proba method is not implemented.")

    def predict_log_proba(self, X):
        """Predict class log-probabilities."""
        raise NotImplementedError("predict_log_proba method is not implemented.")
```

```python
class TransformerMixin:
    """Mixin class for all transformers in mini_sklearn."""

    def fit_transform(self, X, y=None):
        """Fit to data, then transform it."""
        raise NotImplementedError("fit_transform method is not implemented.")

    def transform(self, X):
        """Transform the data."""
        raise NotImplementedError("transform method is not implemented.")
```

```python
class BaseEstimator:
    """Base class for all estimators in mini_sklearn."""

    def __init__(self):
        self._is_fitted = False

    def fit(self, X, y=None):

        """Fit estimator to data."""
        raise NotImplementedError("fit method is not implemented.")

    def score(self, X, y):
        """Return the coefficient of determination R^2 of the prediction."""
        raise NotImplementedError("score method is not implemented.")

    def check_is_fitted(self) -> bool:
        if not self._is_fitted:
            raise Exception('This estimator is not fitted.')
```

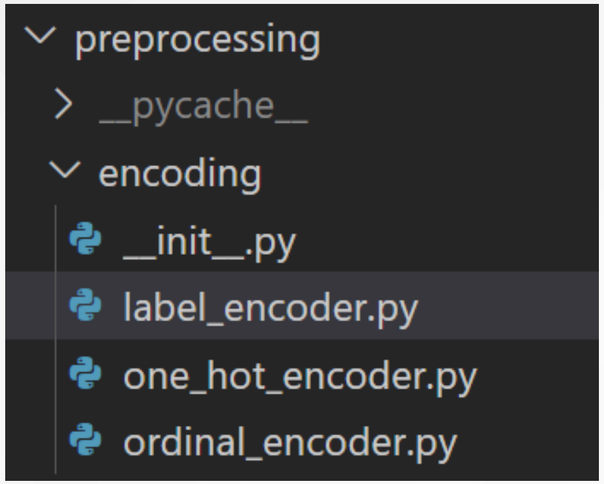
```python
class RegressorMixin:
    """Mixin class for all regressors in mini_sklearn."""

    def score(self, X, y, sample_weight=None):
        """Return the coefficient of determination R^2 of the prediction."""
        y_pred = self.predict(X)

        return r2_score(y, y_pred)
```

Library Exploration

Preprocessing

Preprocessing

ENCODING




```
(TransformerMixin, BaseEstimator):
```

```
def fit(self, y):···

def fit_transform(self, y):···

def transform(self, y):        You, 3 weeks ago
```

Tests

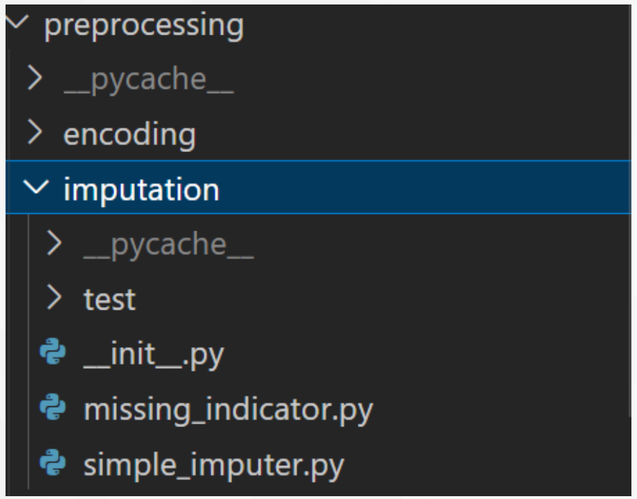
```
mini_sklearn python -m tests.encoding_test.label_encoder
Custom LabelEncoder encoded result:
[1 0 1 2 0 1 2 1]
scikit-learn LabelEncoder encoded result:
[1 0 1 2 0 1 2 1]
Accuracy between custom and sklearn LabelEncoder: 100.0%
```

```
Custom OneHotEncoder result:
[[0. 1. 0. 1. 0.]
 [0. 0. 1. 0. 1.]
 [0. 1. 0. 1. 0.]
 [1. 0. 0. 0. 1.]]
scikit-learn OneHotEncoder result:
[[0. 1. 0. 1. 0.]
 [0. 0. 1. 0. 1.]
 [0. 1. 0. 1. 0.]
 [1. 0. 0. 0. 1.]]
```

```
Custom OrdinalEncoder result:
[[1 0]
 [2 1]
 [1 0]
 [0 1]]
scikit-learn OrdinalEncoder result:
[[1. 0.]
 [2. 1.]
 [1. 0.]
 [0. 1.]]
```

Preprocessing

IMPUTATION




```
(TransformerMixin, BaseEstimator):

    def fit(self, y): ···

    def fit_transform(self, y): ···

    def transform(self, y):        You, 3 weeks ago
```

Simple Imputer

+

Missing Indicator Imputer

Preprocessing

FEATURE SELECTION

Preprocessing

FEATURE SCALING

Preprocessing

NORMALIZATION

Supervised Learning

Supervised Learning

LINEAR MODELS

REGRESSOR

```
class LinearRegression(RegressorMixin, BaseEstimator):
```

CLASSIFIER

```
class LogisticRegression(ClassifierMixin, BaseEstimator):
```

Tests

```
Mean Squared Error of my linear regression model:
2634.8301156270127
Mean Squared Error of sklearn's linear regression
model: 2634.830115627009
```

```
Accuracy score of my logistic regression model: 0.956140350877193
Accuracy score of sklearn's logistic regression model: 0.9298245614
```

Supervised Learning

NEIGHBORS

Supervised Learning

NAIVE BAYES

```python
class BaseNB(BaseEstimator, ClassifierMixin):
    def predict(self, X): ···

    def predict_proba(self, X): ···

    def _encode(self, y): ···
```

```python
class GaussianNB(BaseNB): ···


IlyasIsHere, 9 hours ago | 2 authors (IlyasIsHere and others)
class BernoulliNB(BaseNB): ···


IlyasIsHere, 9 hours ago | 2 authors (IlyasIsHere and others)
class MultinomialNB(BaseNB): ···
```

Tests

```
My Gaussian Naive Bayes Accuracy: 0.9385964912280702
My Bernoulli Naive Bayes Accuracy: 0.4298245614035088
My Multinomial Naive Bayes Accuracy: 0.9122807017543859
Scikit-learn Gaussian Naive Bayes Accuracy: 0.9298245614035088
Scikit-learn Bernoulli Naive Bayes Accuracy: 0.5701754385964912
Scikit-learn Multinomial Naive Bayes Accuracy: 0.9122807017543859
```

Supervised Learning

DECISION TREES

Optimized approach:

Each stump => O(ndlog(n)))

```
You, 4 hours ago | 2 authors (IlyasIsHere and others)
class BaseDecisionTree(BaseEstimator): ···

IlyasIsHere, 9 hours ago | 1 author (IlyasIsHere)
class TreeNode(): ···

You, 3 hours ago | 2 authors (IlyasIsHere and others)
class DecisionTreeClassifier(ClassifierMixin, BaseDecisionTree):

You, 3 hours ago | 2 authors (IlyasIsHere and others)
class DecisionTreeRegressor(RegressorMixin, BaseDecisionTree):
```

Tests

```
Accuracy score of my decision tree classifier: 0.956140350877193
Depth of my decision tree classifier: 6
Accuracy score of sklearn's decision tree classifier: 0.9473684210526315
Depth of sklearn's decision tree classifier: 6
```

```
Mean Squared Error of my decision tree regressor: 0.503683003904312
Depth of my decision tree regressor: 34
Mean Squared Error of scikit-learn's decision tree regressor: 0.4953667177026889
Depth of scikit-learn's decision tree regressor: 34
```

Ensemble Methods

Ensemble Methods

RANDOM FOREST

CLASSIFIER

```
class RandomForestClassifier(ClassifierMixin, BaseEstimator):
    def fit(self, X, y): ···

    def predict_proba(self, X): ···

    def predict(self, X): ···

    @property
    def feature_importances_(self): ···
```

REGRESSOR

```
class RandomForestRegressor(RegressorMixin, BaseEstimator):
    def fit(self, X, y): ···

    def predict(self, X): ···

    @property
    def feature_importances_(self): ···
```

Ensemble Methods

RANDOM FOREST TEST

CLASSIFIER

REGRESSOR

Ensemble Methods

BOOSTING

```
class AdaBoostClassifier(ClassifierMixin, BaseEstimator):
```

```
class GradientBoostingClassifier(ClassifierMixin, BaseEstimator):
```

Ensemble Methods

STACKING

```
class AdaBoostClassifier(ClassifierMixin, BaseEstimator):
```

```
class GradientBoostingClassifier(ClassifierMixin, BaseEstimator):
```

Ensemble Methods

VOTING

```
class AdaBoostClassifier(ClassifierMixin, BaseEstimator):
```

```
class GradientBoostingClassifier(ClassifierMixin, BaseEstimator):
```

Ensemble Methods

STACKING

```
class AdaBoostClassifier(ClassifierMixin, BaseEstimator):
```

```
class GradientBoostingClassifier(ClassifierMixin, BaseEstimator):
```

Neural Networks

Neural Networks

MULTILAYER PERCEPTRON

```
class NeuralNetwork(ClassifierMixin, BaseEstimator):
    def __init__(self, layers, learning_rate=0.01, epochs=1000):···

    def softmax(self, z):···

    def softmax_derivative(self, output, y):···

    def sigmoid(self, z):···

    def sigmoid_derivative(self, z):···
```

```
def one_hot_encode(self, y, num_classes):···

def forward(self, X):···

def backward(self, X, y, activations):···

def fit(self, X, y):···

def predict(self, X):···

def predict_proba(self, X):···
```

Neural Networks

MULTILAYER PERCEPTRON TESTS

Model Selection

Model Selection

GRIDSEARCH

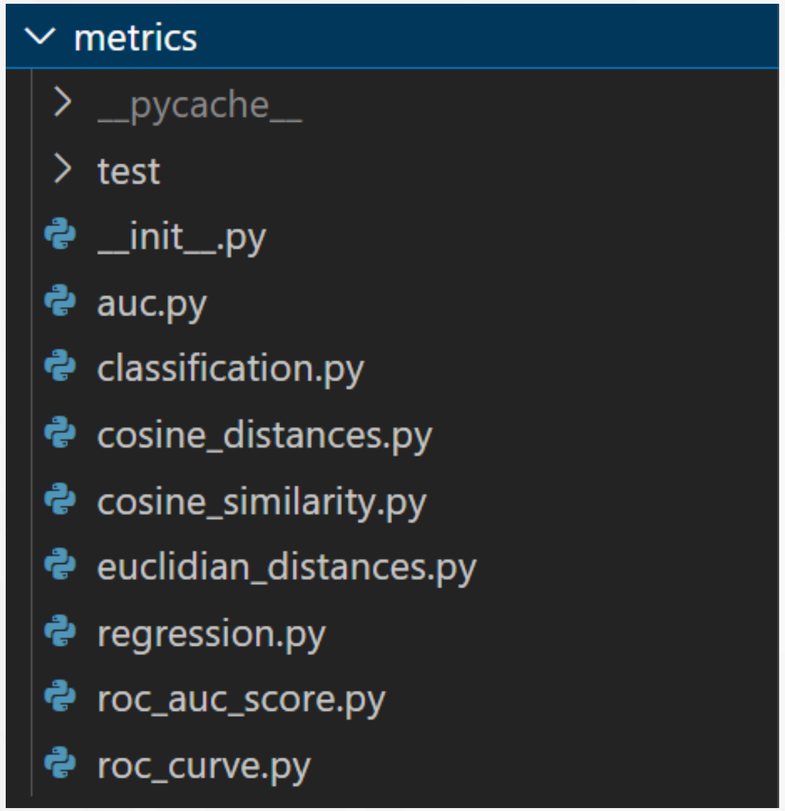
```
class GridSearchCVBase(BaseEstimator):
    def __init__(self, estimator, param_grid):···

    def generate_grid(self):···

    def fit(self, X, y):···

    def get_cv(self, y):···

    def decision_function(self, X):···

    def predict(self, X):···
```

```
class GridSearchCVClassifier(GridSearchCVBase):
    def get_cv(self, y):
        return StratifiedKFold()
```

```
class GridSearchCVRegressor(GridSearchCVBase):
    def get_cv(self, y):
        return KFold()
```

Model Selection

METRICS - TRAIN_TEST_SPLIT - CROSS_VAL




```
class BaseKFold(ABC): ···

You, 3 weeks ago | 1 author (You)
class KFold(BaseKFold): ···


You, 4 hours ago | 1 author (You)
class StratifiedKFold(BaseKFold): ···


def train_test_split(*arrays, test_size=0.25, random_state=None, shuffle=True):
```

```
def cross_val_predict(estimator, X, y, method="predict"): ···
```

Model Selection

METRICS - TRAIN_TEST_SPLIT - CROSS_VAL

Decomposition

Decomposition

PCA

```
class PCA(BaseEstimator, TransformerMixin):
    def __init__(self, n_components=2):···

    def fit(self, X, y=None):···

    def transform(self, X):···

    def inverse_transform(self, X):        You,
```