SKLEARN MINI

A lightweight implementation of ML techniques

By:

- **Sami AGOURRAM**
- **Ilyas HAKKOU**

College of Computing

UM6P University Mohammed VI Polytechnic

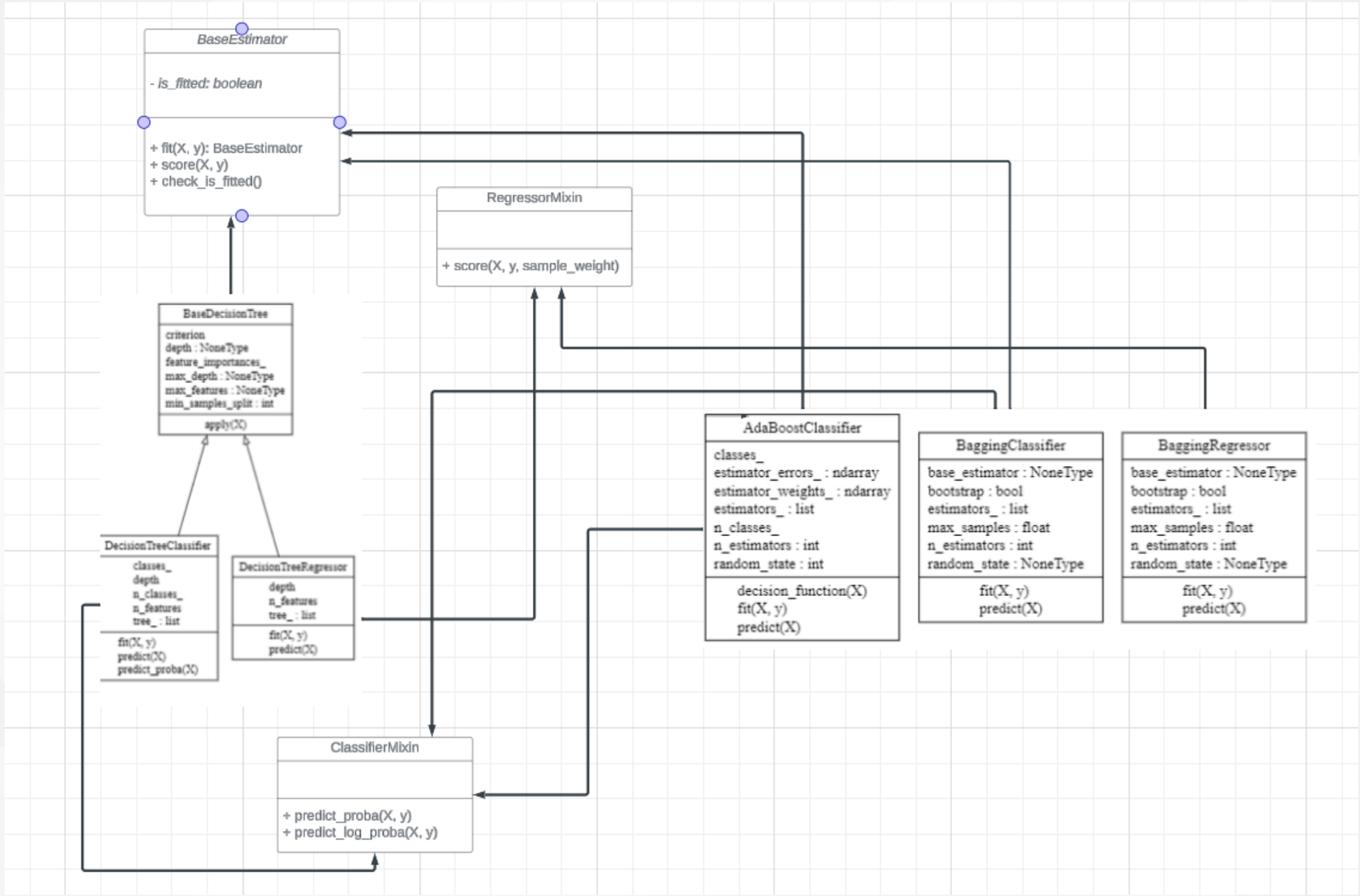
PROJECT OVERVIEW

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.

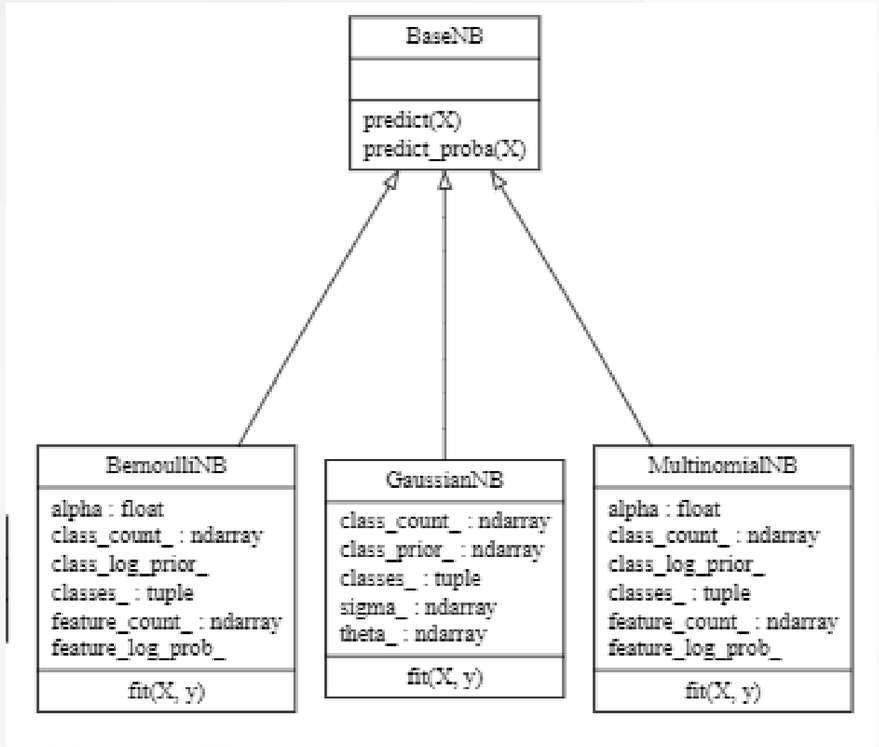
OUTLINE

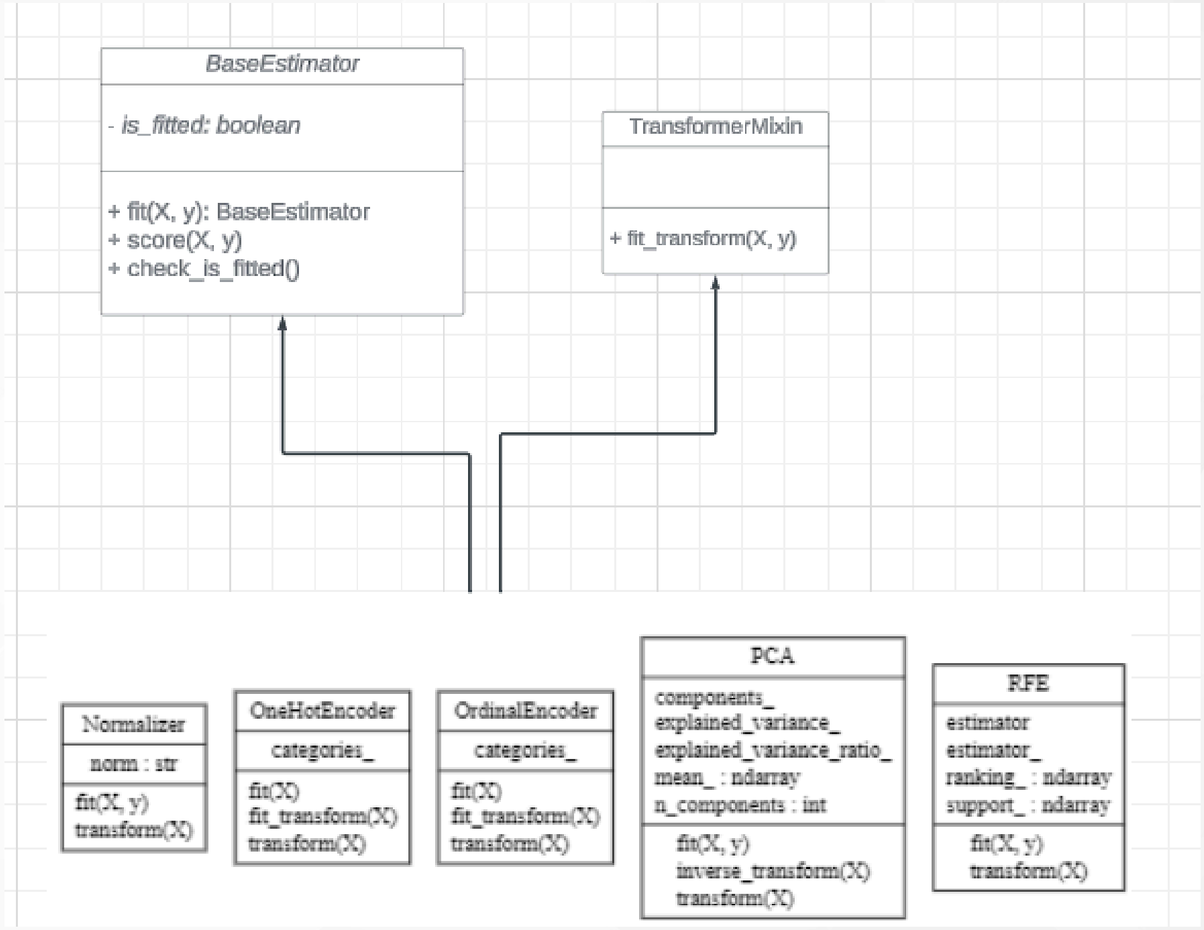
- Project design(Class Diagram, Library architecture)
- Library exploration :
  - Preprocessing :
    - Feature Scaling
    - Normalization
    - Imputation
    - Encoding
    - Feature Selection
  - Supervised Learning :
    - Linear Models
    - Neighbors
    - Naive Bayes
    - Decision Trees
  -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

PROJECT DESIGN



PROJECT DESIGN

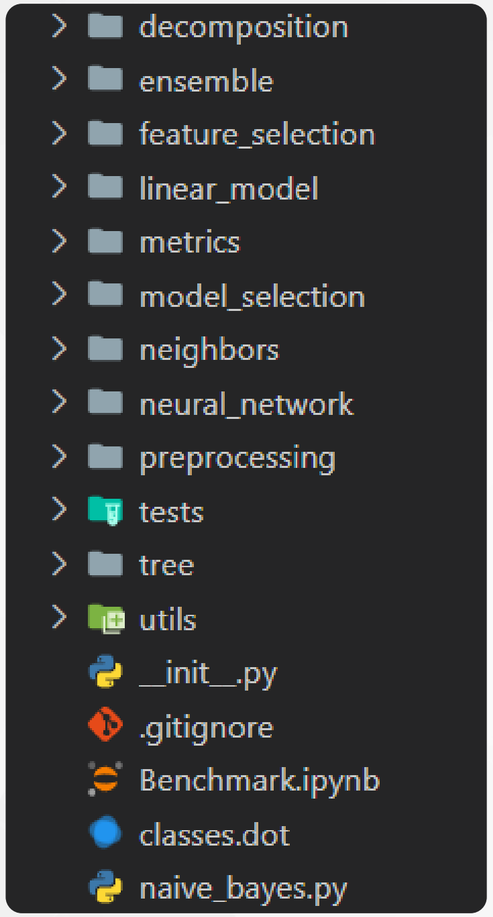


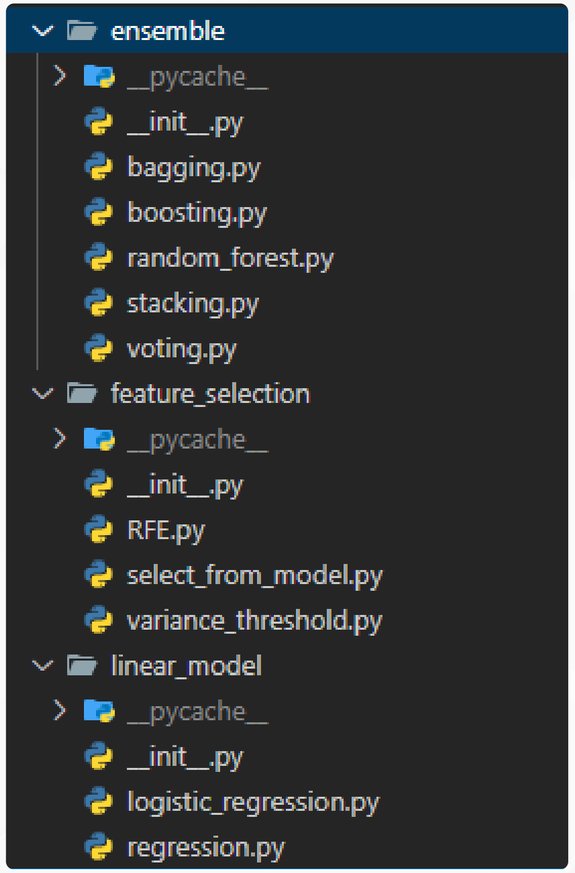


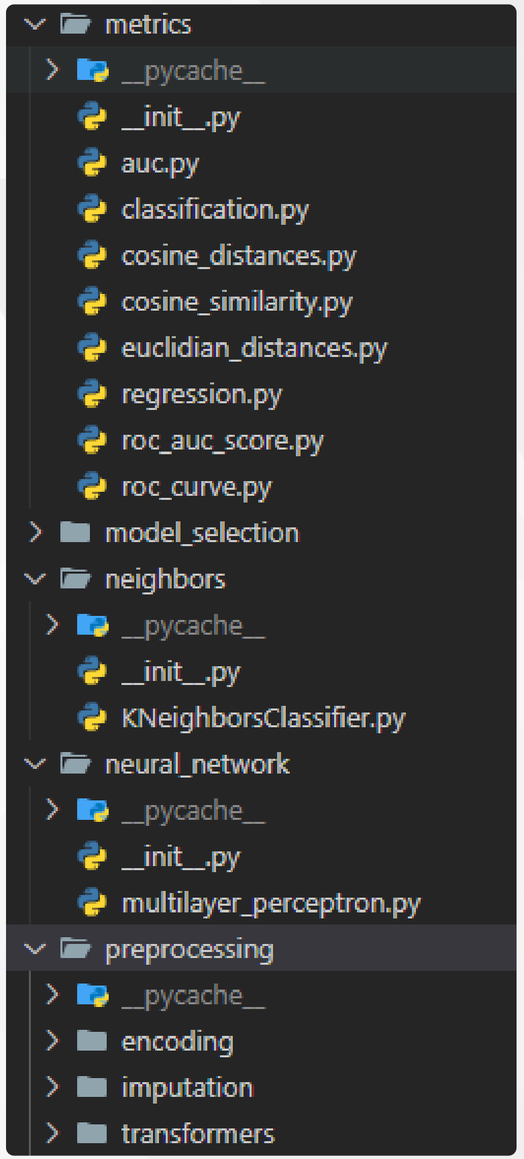
PACKAGE ON PIP

pip install um6p-CC-learn

LIBRARY ARCHITECTURE







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.')
```

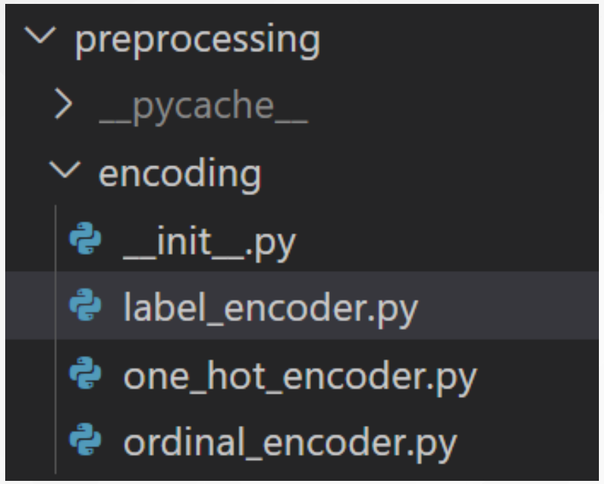
```python
class RegressorMixin:
    """Mixin class for all regressors in mini_sklearn."""

    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

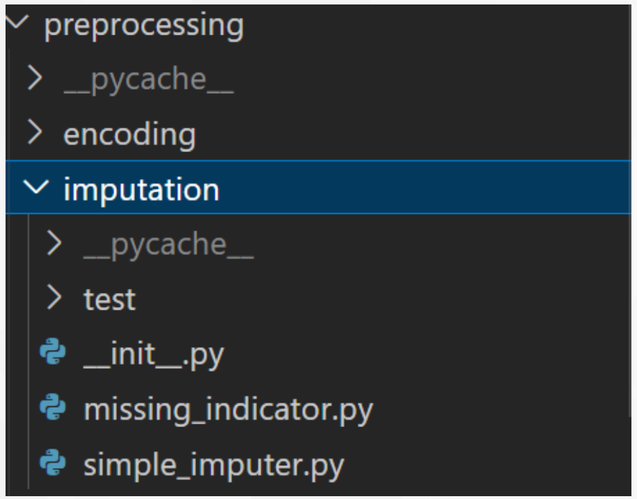
```
mini_sklearn python -m tests.encoding_test.label_encoder
Custom LabelEncoder encoded result:
[1 0 1 2 0 1 2 1]
scikit-learn LabelEncoder encoded result:
[1 0 1 2 0 1 2 1]
Accuracy between custom and sklearn LabelEncoder: 100.0%
```

```
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

```
class RFE(TransformerMixin, BaseEstimator):
    """Feature ranking with recursive feature elimination (RFE).

    This transformer recursively removes features, building a model on the
    remaining features at each step until the desired number of features is
    reached.
```

```
class SelectFromModel(TransformerMixin, BaseEstimator):
    """Meta-transformer for selecting features based on importance weights.

    This transformer selects features based on the importance weights
    provided by an estimator. Features whose importance exceeds a given
    threshold are retained, while others are discarded.
```

```
class VarianceThreshold(TransformerMixin, BaseEstimator):
    """Feature selector that removes low-variance features.

    This transformer removes features whose variance does not meet a certain
    threshold. By default, it removes features with zero variance.
```

Preprocessing

Feature scaling

```
class StandardScaler(TransformerMixin, BaseEstimator):
    """Standardize features by removing the mean and scaling to unit variance.
```

```
class MinMaxScaler(TransformerMixin, BaseEstimator):
    """Transforms features by scaling each feature to a given range.
```

Preprocessing

Normalization

```
class Normalizer(TransformerMixin, BaseEstimator):
    """
    Normalize samples individually to unit norm.

    This transformer normalizes each sample to have unit norm.
    The 'norm' parameter specifies the type of normalization to apply:
    - 'l1': L1 normalization (sum of absolute values)
    - 'l2': L2 normalization (Euclidean norm)
    - 'max': Max normalization (maximum absolute value)
```

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: KNN

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

Tests

```
My KNeighborsClassifier
Accuracy: 0.9806678383128296
Precision: 0.9726775956284153
Recall: 0.9971988795518207
F1-score: 0.9847856154910096
Time taken to fit: 0.0
```

```
Scikit-learn's KNeighborsClassifier
Accuracy: 0.9806678383128296
Precision: 0.9726775956284153
Recall: 0.9971988795518207
F1-score: 0.9847856154910096
Time taken to fit: 0.0009992122650146484
```

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): ···
```

Random forest regressor + classifier

CLASSIFIER

```
Time taken to fit my random forest classifier: 274.24708557128906
 seconds
Time taken to fit scikit-learn's random forest classifier: 0.1360
313892364502 seconds
Accuracy score of my random forest classifier: 0.9649122807017544
Accuracy score of scikit-learn's random forest classifier: 0.9561
```

```
Time taken to fit my random forest classifier with max_depth=10:
135.2256429195404 seconds
Time taken to fit scikit-learn's random forest classifier with ma
x_depth=10: 0.06699991226196289 seconds
Accuracy score of my random forest classifier with max_depth=10:
0.9649122807017544
Accuracy score of scikit-learn's random forest classifier with ma
x_depth=10: 0.956140350877193
```

REGRESSOR

```
Time taken to fit my random forest regressor:
333.5051608085632 3 seconds
Time taken to fit scikit-learn's random forest regressor:
0.4485094545727172 85 seconds
Mean Squared Error of my random forest regressor:
0.4411786211511665
Mean Squared Error of scikit-learn's random forest regressor:
0.4399402399163459
```

Ensemble Methods

Boosting:

- AdaBoost
- GradientBoosting

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

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

Tests

```
Accuracy score of my AdaBoost classifier: 0.903508771
Accuracy score of sklearn's AdaBoost classifier: 0.94
```

```
Accuracy of my GradientBoostingClassifier: 0.956140350877193
Accuracy of scikit-learn's GradientBoostingClassifier: 0.956140350877193
```

Stacking: Classifier + Regressor

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

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

Voting: Classifier + Regressor

2 options for classifier: ‘hard’ and ‘soft’

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

```
My Soft Voting Classifier Accuracy: 0.9933333333333333
Sklearn's Soft Voting Classifier Accuracy: 0.9933333333333333
My Hard Voting Classifier Accuracy: 1.0
Sklearn's Hard Voting Classifier Accuracy: 1.0
```

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

```
Voting Regressor R2 Score (My Model): 0.860251390246441
Voting Regressor R2 Score (Sklearn's Model): 0.860251390246441
--------------------------------------------------
Test passed!
```

Bagging: Classifier + Regressor

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

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

Tests

```
Custom Bagging Classifier Accuracy: 1.0
sklearn Bagging Classifier Accuracy: 1.0
Custom Bagging Regressor MSE: 0.036056356022765 14
sklearn Bagging Regressor MSE: 0.036301980126523054
```

Neural Networks

Neural Networks

Multilayer Perceptron (MLP Classifier)

```
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):···
```

Multilayer perceptron tests

```
Custom Neural Network
- Fit Time: 1.0839s,
Accuracy: 1.0000,
Precision: 1.0000,
Recall: 1.0000,
F1 Score: 1.0000
```

```
Scikit-learn Neural Network
- Fit Time: 0.0870s,
Accuracy: 1.0000,
Precision: 1.0000,
Recall: 1.0000,
F1 Score: 1.0000
```

Model Selection

Model Selection

Grid Search

```python
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):...
```

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

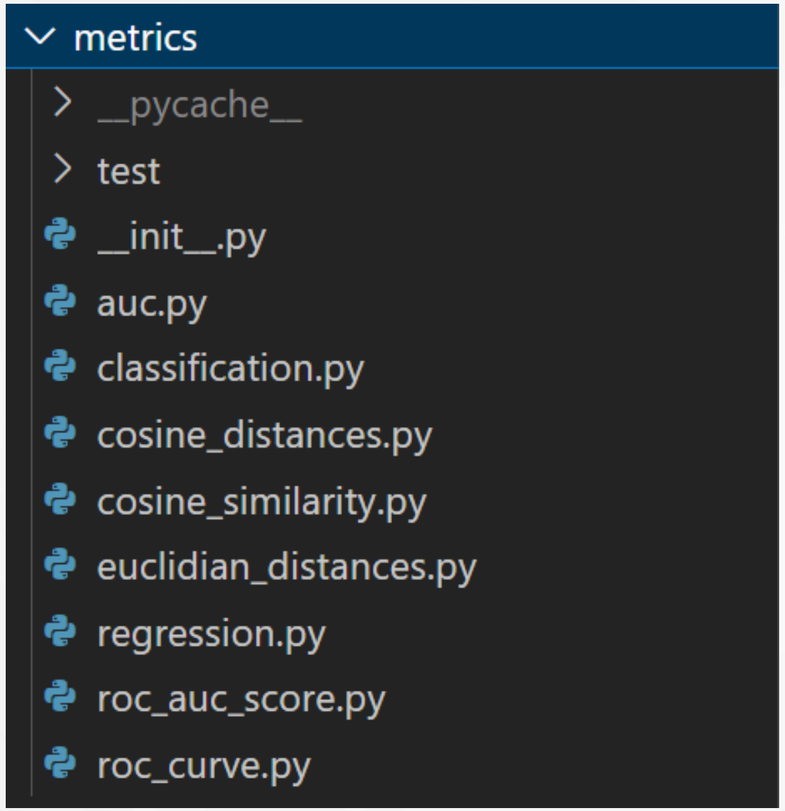
```python
class GridSearchCVRegressor(GridSearchCVBase):
    def get_cv(self, y):
        return KFold()
```

Tests

```
Classification Comparison:
Custom GridSearchCV Best Params: {'C': 1, 'kernel': 'linear'}
Sklearn GridSearchCV Best Params: {'C': 1, 'kernel': 'linear'}
Custom GridSearchCV Best Score: 0.9800000000000001
Sklearn GridSearchCV Best Score: 0.9800000000000001

Regression Comparison:
Custom GridSearchCV Best Params: {'C': 10, 'kernel': 'rbf'}
Sklearn GridSearchCV Best Params: {'C': 10, 'kernel': 'rbf'}
Custom GridSearchCV Best Score: 0.4724614915908951
Sklearn GridSearchCV Best Score: 0.4724614915908951
```

Model Selection

Metrics - train_test_split - cross_val



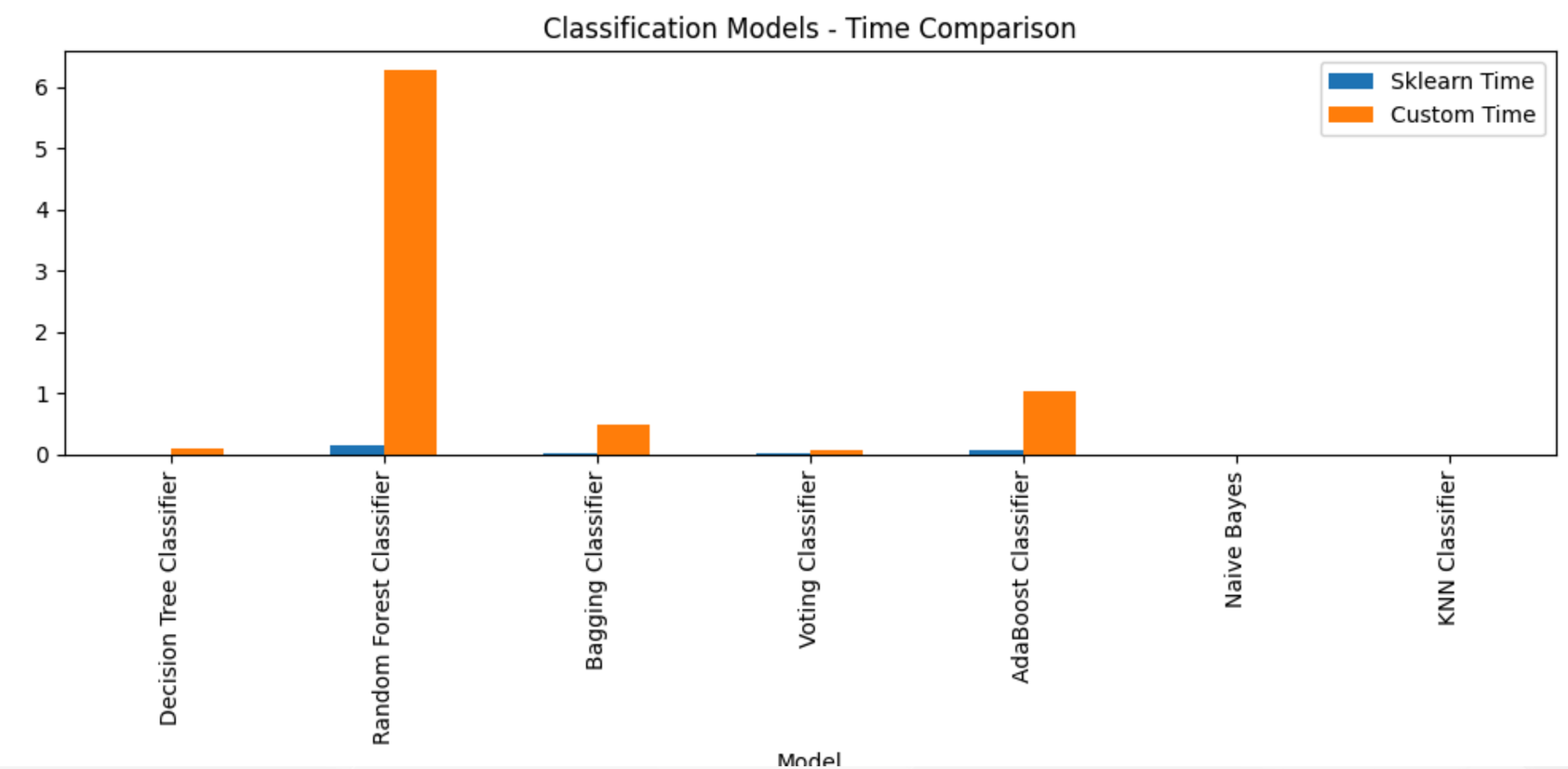

```python
class BaseKFold(ABC):···

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):
```

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

Benchmark

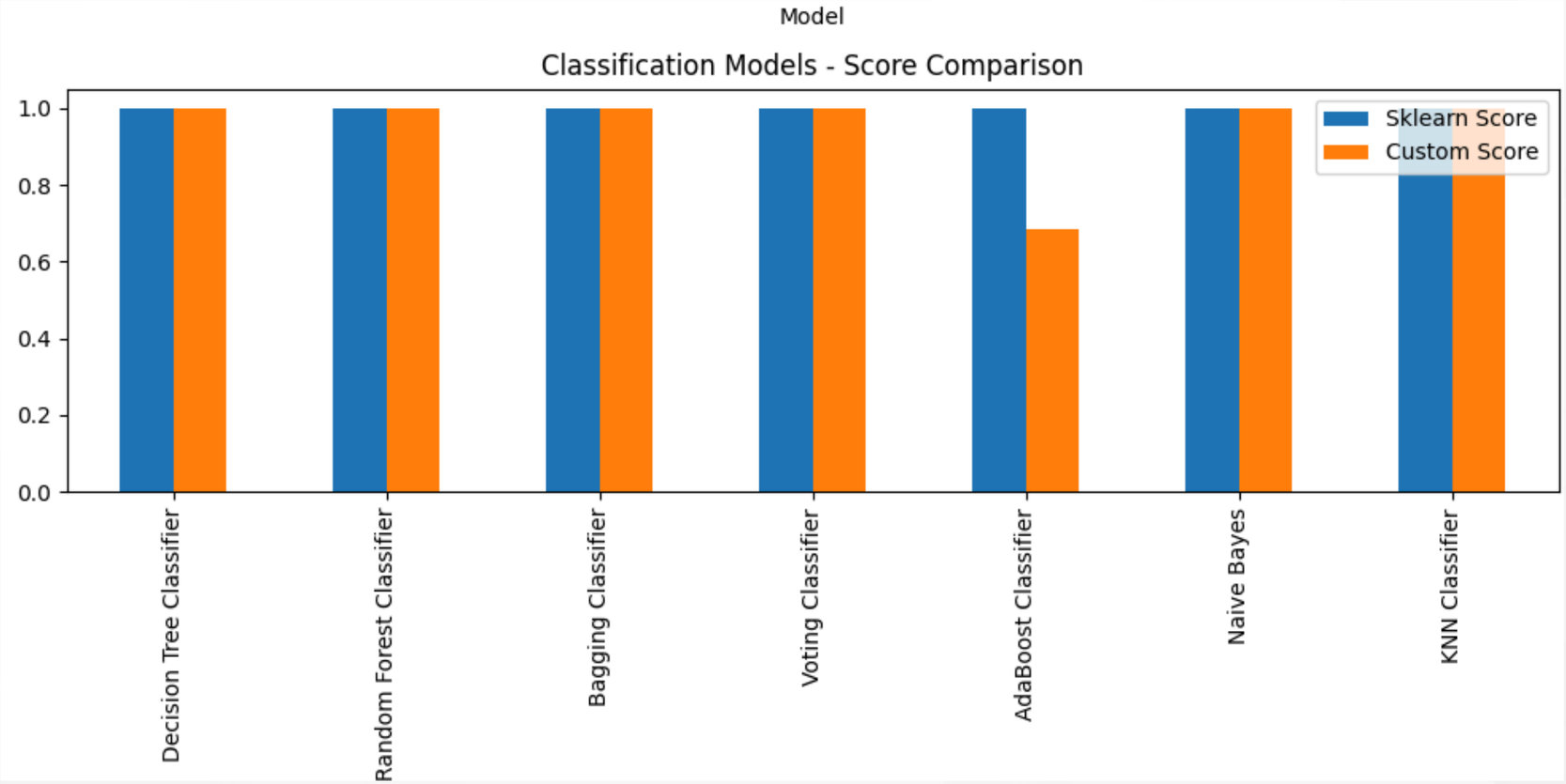
Benchmarks

Classifiers fitting time



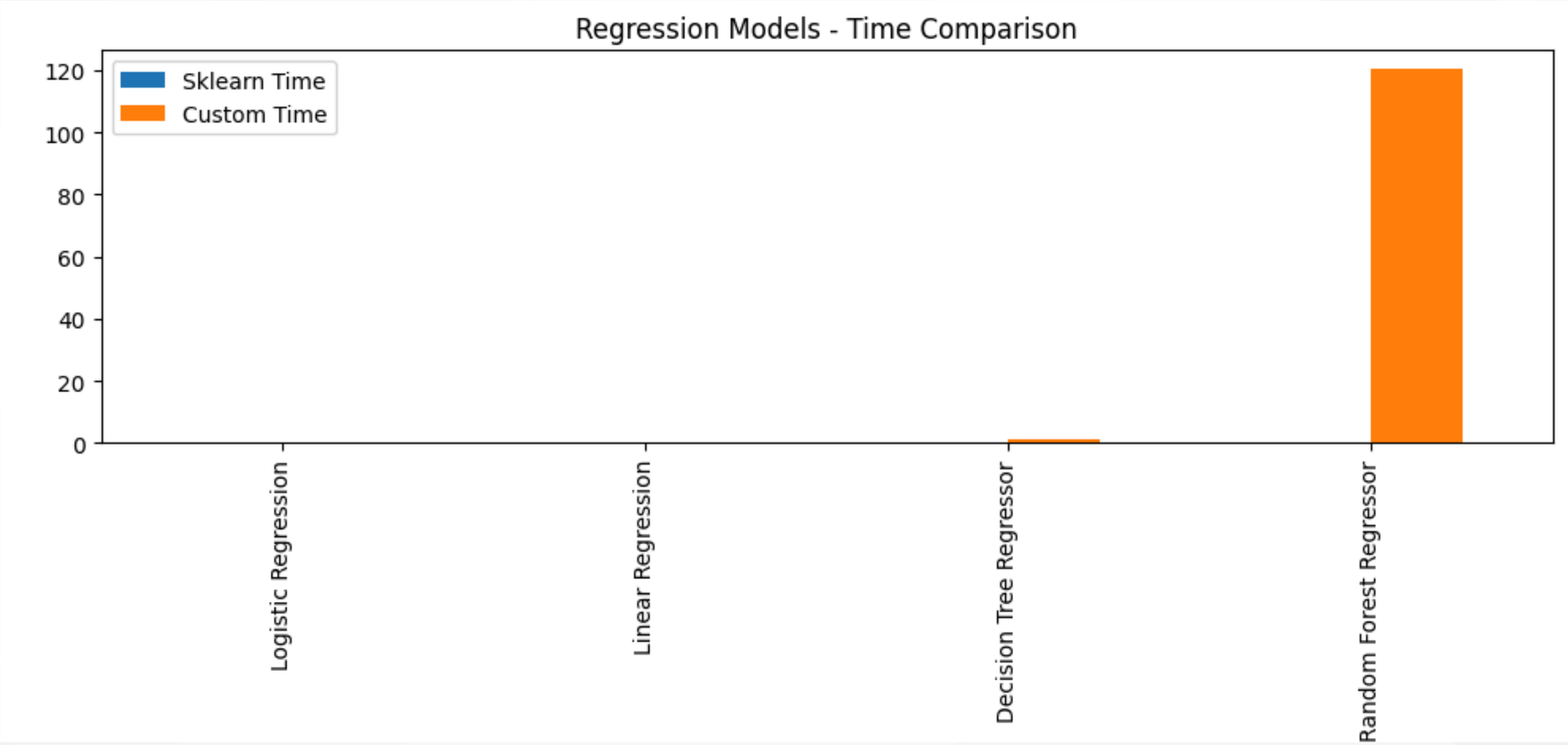
Benchmarks

Classifiers scores



Benchmarks

Regressors fitting time



Benchmarks

Regressors scores (MSE)