Project Report: Mini-Scikit-Learn Implementation

List of Implemented Modules and Their Content

ensemble

- `AdaBoost.py`
  - **AdaBoost** - Adaptive Boosting implementation using DecisionTreeClassifier as the base estimator.
- `GradientBoostingClassifier.py`
  - **GradientBoostingClassifier** - Gradient Boosting Classifier using DecisionTreeRegressor as the base estimator.
- `GradientBoostingRegressor.py`
  - **GradientBoostingRegressor** - Gradient Boosting Regressor using DecisionTreeRegressor as the base estimator.
- `RandomForestClassifier.py`
  - **RandomForestClassifier** - Random Forest classifier.
- `RandomForestRegressor.py`
  - **RandomForestRegressor** - Random Forest regressor.
- `StackingClassifier.py`
  - **StackingClassifier** - Stacking Classifier implementation using base models and a meta-model for ensemble learning.
- `VotingClassifier.py`
  - **VotingClassifier** - Voting Classifier implementation that allows for both hard and soft voting.

metrics

- `Accuracy.py`
  - **Accuracy** - Metric for calculating accuracy.
- `BaseMetric.py`
  - **BaseMetric** - Base class for all metrics.
- `ConfusionMatrix.py`
  - **ConfusionMatrix** - Metric for calculating confusion matrix.
- `F1Score.py`
  - **F1Score** - Metric for calculating F1 score.
- `MeanAbsoluteError.py`
  - **MeanAbsoluteError** - Metric for calculating mean absolute error.
- `MeanSquaredError.py`
  - **MeanSquaredError** - Metric for calculating mean squared error.
- `Precision.py`
  - **Precision** - Metric for calculating precision.
- `Recall.py`
  - **Recall** - Metric for calculating recall.
- `RootMeanSquaredError.py`
  - **RootMeanSquaredError** - Metric for calculating root mean squared error.
- `RSquared.py`
  - **RSquared** - Metric for calculating R squared.

model_selection

- `GridSearchCV.py`
  - **GridSearchCV** - Exhaustive search over specified parameter values for an estimator.
- `KFold.py`
  - **KFold** - K-Folds cross-validator.

- `ParameterGrid.py`
  - **ParameterGrid** - Grid of parameters with a discrete number of values for each parameter.
- `train_test_split.py`
  - **train_test_split** - Split arrays or matrices into random train and test subsets with optional stratification.

neural_networks

- `MLP.py`
  - **MLP** - Multi-Layer Perceptron for classification tasks.
- `MLPRegressor.py`
  - **MLPRegressor** - Multi-Layer Perceptron for regression tasks.
- `Perceptron.py`
  - **Perceptron** - Perceptron classifier.

preprocessing

- `LabelEncoder.py`
  - **LabelEncoder** - Encode target labels with value between 0 and n_classes-1.
- `MinMaxScaler.py`
  - **MinMaxScaler** - Scaling features to a given range, usually [0, 1].
- `OneHotEncoder.py`
  - **OneHotEncoder** - Encode categorical integer features as a one-hot numeric array.
- `SimpleImputer.py`
  - **SimpleImputer** - Imputation transformer for completing missing values.
- `StandardScaler.py`
  - **StandardScaler** - Standardize features by removing the mean and scaling to unit variance.

supervised_learning

classification

- `DecisionTreeClassifier.py`
  - **DecisionTreeClassifier** - Decision Tree classifier.
- `KNNClassifier.py`
  - **KNNClassifier** - K-Nearest Neighbors classifier.
- `LogisticRegression.py`
  - **LogisticRegression** - Logistic Regression classifier.
- `NaiveBayes.py`
  - **NaiveBayes** - Naive Bayes classifier.
- `SVM.py`
  - **SVM** - Support Vector Machine classifier.

regression

- `DecisionTreeRegressor.py`
  - **DecisionTreeRegressor** - Decision Tree regressor.

utilities

- `clone.py`
  - **clone** - Utility to clone an estimator.

Test

- `GlobalTest.ipynb` - Global tests for the entire library.
- `GradientBoostingClassifierTest.ipynb` - Tests for GradientBoosting-Classifier.
- `GradientBoostingRegressorTest.ipynb` - Tests for GradientBoostingRegressor.
- `PerceptronTest.ipynb` - Tests for Perceptron.

Code Documentation

All modules and classes are documented with docstrings that describe their functionality, parameters, and methods. These docstrings are formatted to be compatible with documentation generation tools like Sphinx.