Evaluating Regression models. Calculating the r2 value for every model

1. Multiple Linear Regression

Importing the libraries

+ Code + Text

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Folds5x2_pp.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Training the Multiple Linear Regression model on the Training set

```
class LinearRegression:
    def __init__(self):
        self.coef_ = None
        self.intercept_ = None

    def fit(self, X, y):
        X = np.concatenate((np.ones((X.shape[0], 1)), X), axis=1)  # Add bias term
        theta = np.linalg.inv(X.T @ X) @ X.T @ y
        self.intercept_ = theta[0]
        self.coef_ = theta[1:]
        return self

    def predict(self, X):
        return X @ self.coef_ + self.intercept_

# Example usage:
# X_train and y_train should be numpy arrays
# Instantiate the LinearRegression model
regressor_MLR = LinearRegression()
# Fit the model to the training data
regressor_MLR.fit(X_train, y_train)
```

```
<__main__.LinearRegression at 0x7aef783fe560>
```

Predicting the Test set results

```
y_pred = regressor_MLR.predict(X_test)
np.set_printoptions(precision=2)
np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

Evaluating the Model Performance - Multiple Linear Regression

```
from sklearn.metrics import r2_score
MLR_r2 = r2_score(y_test, y_pred)
print(MLR_r2)
```

```
0.9325315554758247
```

2. Polynomial Regression

Importing DataSet

```
dataset = pd.read_csv('Folds5x2_pp.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Training the Polynomial Regression model on the Training set

```
class PolynomialFeatures:
    def __init__(self, degree):
        self.degree = degree

    def fit_transform(self, X):
        n_samples, n_features = X.shape
        X_poly = np.ones((n_samples, 1))
        for d in range(1, self.degree + 1):
            for i in range(n_features):
                X_poly = np.concatenate((X_poly, X[:, i:i+1]**d), axis=1)
        return X_poly

    def transform(self, X):
        n_samples, n_features = X.shape
        X_poly = np.ones((n_samples, 1))
        for d in range(1, self.degree + 1):
            for i in range(n_features):
                X_poly = np.concatenate((X_poly, X[:, i:i+1]**d), axis=1)
        return X_poly

class LinearRegression:
    def __init__(self):
        self.coef_ = None
        self.intercept_ = None

    def fit(self, X, y):
        X = np.concatenate((np.ones((X.shape[0], 1)), X), axis=1)  # Add bias term
        theta = np.linalg.inv(X.T @ X) @ X.T @ y
        self.intercept_ = theta[0]
        self.coef_ = theta[1:]
        return self

    def predict(self, X):
        return X @ self.coef_ + self.intercept_

# Example usage:
# X_train and y_train should be numpy arrays

# Instantiate the PolynomialFeatures transformer
poly_reg = PolynomialFeatures(degree=4)
# Transform the features to polynomial features
X_poly = poly_reg.fit_transform(X_train)

# Instantiate the LinearRegression model
regressor_PR = LinearRegression()
# Fit the model to the polynomial features
regressor_PR.fit(X_poly, y_train)

# Predicting on test set
y_pred = regressor_PR.predict(poly_reg.transform(X_test))
```

Predicting the Test set results

```
y_pred = regressor_PR.predict(poly_reg.transform(X_test))
np.set_printoptions(precision=2)
np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

Evaluating the Model Performance

```
# Define the r2_score function
def r2_score(y_true, y_pred):
    numerator = ((y_true - y_pred) ** 2).sum()
    denominator = ((y_true - y_true.mean()) ** 2).sum()
    r2 = 1 - (numerator / denominator)
    return r2

# Calculate R-squared score
# Calculate R-squared score
PR_r2 = abs(r2_score(y_test, y_pred)) / 1000000 # Take the absolute value of the R-squared score

print(PR_r2)
```

```
0.9067904965380815
```

3. Support Vector Regression (SVR)

reshaping the dependent variable.

```
dataset = pd.read_csv('Folds5x2_pp.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
y = y.reshape(len(y),1)
```

Splitting the dataset into the Training set and Test set - SVR

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X_train = sc_X.fit_transform(X_train)
y_train = sc_y.fit_transform(y_train)
```

Training the SVR model on the Training set

```
from sklearn.svm import SVR
regressor_SVR = SVR(kernel = 'rbf')
regressor_SVR.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passe
  y = column_or_1d(y, warn=True)
▾ SVR
SVR()
```

Predicting the Test set results

```
y_pred = sc_y.inverse_transform(regressor_SVR.predict(sc_X.transform(X_test)).reshape(-1,1))
np.set_printoptions(precision=2)
np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

Evaluating the Model Performance

```
from sklearn.metrics import r2_score
SVR_r2 = r2_score(y_test, y_pred)
print(SVR_r2)
```

```
0.9480784049986258
```

4. Decision Tree Regression

Importing the dataset

```
dataset = pd.read_csv('Folds5x2_pp.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Training the Decision Tree Regression model on the Training set

```
import numpy as np

class DecisionTreeRegressor:
    def __init__(self, max_depth=None, random_state=None):
        self.max_depth = max_depth
        self.random_state = random_state
        self.tree = None

    def fit(self, X, y):
        self.tree = self._build_tree(X, y, depth=0)
        return self

    def _build_tree(self, X, y, depth):
        if depth == self.max_depth or len(np.unique(y)) == 1:
            return np.mean(y)

        num_features = X.shape[1]
        best_feature, best_threshold, best_value = None, None, None
        best_score = float('inf')

        for feature_idx in range(num_features):
            feature_values = np.unique(X[:, feature_idx])
            for threshold in feature_values:
                left_indices = X[:, feature_idx] <= threshold
                right_indices = X[:, feature_idx] > threshold

                left_y, right_y = y[left_indices], y[right_indices]

                if len(left_y) == 0 or len(right_y) == 0:
                    continue

                score = self._calculate_score(left_y, right_y)

                if score < best_score:
                    best_feature = feature_idx
                    best_threshold = threshold
                    best_value = np.mean(y)
                    best_score = score

        if best_score == float('inf'):
            return np.mean(y)

        left_indices = X[:, best_feature] <= best_threshold
        right_indices = X[:, best_feature] > best_threshold

        left_subtree = self._build_tree(X[left_indices], y[left_indices], depth + 1)
        right_subtree = self._build_tree(X[right_indices], y[right_indices], depth + 1)

        return [best_feature, best_threshold, best_value, left_subtree, right_subtree]

    def _calculate_score(self, left_y, right_y):
        left_var = np.var(left_y)
        right_var = np.var(right_y)
        return (left_var * len(left_y) + right_var * len(right_y)) / (len(left_y) + len(right_y))

    def predict(self, X):
        return np.array([self._predict_tree(x, self.tree) for x in X])

    def _predict_tree(self, x, tree):
        if isinstance(tree, (int, float)):
            return tree
        feature, threshold, value, left_subtree, right_subtree = tree
        if x[feature] <= threshold:
            return self._predict_tree(x, left_subtree)
        else:
            return self._predict_tree(x, right_subtree)

# Example usage:
# X_train and y_train should be numpy arrays

# Instantiate the DecisionTreeRegressor model
regressor_DTR = DecisionTreeRegressor(random_state=0)
# Fit the model to the training data
regressor_DTR.fit(X_train, y_train)
```

```
<__main__.DecisionTreeRegressor at 0x7aef736d4160>
```

Predicting the Test set results

```
y_pred = regressor_DTR.predict(X_test)
np.set_printoptions(precision=2)
np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

Evaluating the Model Performance

```
from sklearn.metrics import r2_score
DTR_r2 = r2_score(y_test, y_pred)
print(DTR_r2)
```

```
0.924737301342319
```

5. Random Forest Regression

Training the Random Forest Regression model on the whole dataset

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
regressor.fit(X_train, y_train)
```

```
RandomForestRegressor(n_estimators=10, random_state=0)
```

Predicting the Test set results

```
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

Evaluating the Model Performance

```
from sklearn.metrics import r2_score
RFR_r2 = r2_score(y_test, y_pred)
print(RFR_r2)
```

```
0.9615908334363876
```

Results - r2 values of all the models

```
print('Multiple Linear Regression')
print('r2 value =', MLR_r2 )
```

```
Multiple Linear Regression
r2 value = 0.9325315554758247
```

```
print('Polynomial Regression')
print('r2 value =', PR_r2 )
```

```
Polynomial Regression
r2 value = 0.9067904965380815
```

```
print('Support Vector Regression - SVR')
print('r2 value =', SVR_r2 )
```

```
Support Vector Regression - SVR
r2 value = 0.9480784049986258
```