Q1. What is Gradient Boosting Regression?

Gradient Boosting Regression is a popular machine learning technique used for regression problems. It is an ensemble learning method that combines multiple weak regression models to create a strong predictive model.

In gradient boosting, a series of weak models are trained iteratively, with each subsequent model attempting to correct the errors of the previous model. The basic idea is to fit a simple regression model to the data, then compute the difference between the predicted values and the actual values. The next model then fits to the residuals (i.e., the differences between the predicted and actual values), with the goal of reducing the remaining error.

The "gradient" in gradient boosting refers to the use of gradient descent optimization to minimize the loss function. The loss function used in gradient boosting is typically a measure of the difference between the predicted and actual values, such as mean squared error or mean absolute error.

Gradient boosting can be used with a variety of regression models as base learners, such as decision trees or linear regression models. It is a powerful technique that has been shown to achieve state-of-the-art results on a wide range of regression problems, including prediction of housing prices, stock prices, and more

Q2. Implement a simple gradient boosting algorithm from scratch using Python and NumPy. Use a simple regression problem as an example and train the model on a small dataset. Evaluate the model's performance using metrics such as mean squared error and R-squared.

In [2]:
```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error, r2_score

X, y = make_regression(n_samples=100, n_features=1, noise=10, random_state


split = int(len(X)*0.8)
X_train, y_train = X[:split], y[:split]
X_test, y_test = X[split:], y[split:]

class GradientBoostingRegressor:
    def __init__(self, n_estimators, learning_rate):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.models = []

    def fit(self, X, y):
        # initialize the predictions
        y_pred = np.zeros(len(y))


        for i in range(self.n_estimators):

            gradient = y - y_pred


            model = DecisionTreeRegressor(max_depth=1)
            model.fit(X, gradient)


            self.models.append(model)

            y_pred += self.learning_rate * model.predict(X)

    def predict(self, X):

        y_pred = np.zeros(len(X))

        for model in self.models:
            y_pred += self.learning_rate * model.predict(X)

        return y_pred
from sklearn.tree import DecisionTreeRegressor

gb = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1)
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

```
Mean Squared Error: 67.71525820905791
R-squared Score: 0.9575588436634127
```

Q3. Experiment with different hyperparameters such as learning rate, number of trees, and tree depth to optimise the performance of the model. Use grid search or random search to find the best hyperparameters

In [4]:

```

import numpy as np
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

X, y = make_regression(n_samples=1000, n_features=10, noise=10, random_sta
```

In [5]:

```
split = int(len(X)*0.8)
X_train, y_train = X[:split], y[:split]
X_test, y_test = X[split:], y[split:]


param_grid = {
    'n_estimators': [10,20,30,40,50],
    'learning_rate': [0.01, 0.01, 1],
    'max_depth': [1, 2, 3]
}


gb = GradientBoostingRegressor()


grid_search = GridSearchCV(estimator=gb, param_grid=param_grid, cv=5)


grid_search.fit(X_train, y_train)
```

Out[5]:

```
▸          GridSearchCV
▸ estimator: GradientBoostingRegressor
    ▸ GradientBoostingRegressor
```

In [6]:
```
best_params = grid_search.best_params_

gb = GradientBoostingRegressor(**best_params)


gb.fit(X_train, y_train)
GradientBoostingRegressor(learning_rate=1, max_depth=1, n_estimators=50)
y_pred = gb.predict(X_test)
y_pred
```

```
Out[6]: array([ -30.95323312,  -79.39380921,   65.49935119, -181.05634672,
              -100.29307511,  -20.4359374 , -164.63941687,   71.00112182,
               -51.7829512 ,  -60.71661252, -138.12639865, -207.34729264,
              -271.7140004 ,   25.22420287,  -85.82697331,    6.97388455,
                86.74184225,  133.37624786,   71.14456023,  -18.10225891,
                -2.56859274,   60.55991405,   91.2662676 ,  -82.13616056,
               141.88245043,  -21.65343766,  -29.59597427,  176.95329294,
               -85.99145152,  -17.51643494, -227.00742466, -144.11521404,
               -78.63504533, -181.85805733, -239.7887483 ,   29.543985  ,
              -110.97051887, -135.62188677,   79.86033693,  183.40055536,
               -25.86953995,   36.17126283, -144.80393852,   19.8386141 ,
                35.49998298, -315.11796203,   49.794025  ,  -72.63638294,
                33.56144382,   55.19465723,  199.34296487,  127.97857209,
                77.80963956,   83.15565606,   10.41301613,  -54.50619909,
                44.63783562,  -87.98283721,   62.42099233,   29.8522689 ,
              -139.42159365,   80.52159117,  -52.79338399,  356.82312278,
               238.53968794,  164.86832705,    1.84540863,    5.37366267,
                86.38653944, -206.86022965,   71.96635029,  -66.55051287,
                83.97315976,  106.22755591,  -38.87812511,  -43.10253198,
               -10.32238559,  -81.82483951, -198.45508673,  134.25238013,
              -239.5791962 ,   14.97760004,  -55.25100934,  115.08994903,
                13.1540767 ,   72.68167283,  -68.5296675 ,   90.68412441,
               140.8869464 ,  144.40952525, -256.92398992,   -6.3744315 ,
              -233.77915086,  173.63946751,   33.88447515,  -90.84745517,
               124.31244426, -228.93719156,  -99.51175792,  -99.3427943 ,
               -79.3494139 ,   25.3869847 ,   99.97191838,   36.1443649 ,
               157.4691021 , -168.91487836,  -22.21675468,  219.63517665,
              -208.96280545,   50.30469999,   18.71787851,  -18.92537321,
                83.2392817 ,  -32.12854143, -116.36236928,   -2.04698514,
               -22.180952  ,   -4.16308684, -123.42074301, -195.82560987,
               -52.25820223,  -69.96011225, -127.70552828,  -37.61659479,
              -134.92698567,   46.18943294,  -22.92747861, -102.4011998 ,
                17.47334765,  -91.1132984 ,  -12.99731022,  148.53911441,
                50.06874876,  -10.594364  ,  -19.46548149,  180.64161398,
                62.76739062,  226.92489354,  175.09683168,  -41.04751801,
               283.94483474,   79.12160882,  -39.2432803 ,  -11.42738906,
               -52.6292072 ,    7.47549562, -128.12022957,  263.34958951,
               233.89053854,   55.17631714,   76.73236555,  298.97457797,
                29.68991132,  218.94725394,  -56.46214453,  168.99361441,
               114.56263676, -140.65485961,  101.01363864,  -43.71704123,
                38.71484609,  137.83713615,   46.28520494, -185.93140072,
               107.21197638,  126.93897787, -294.76248471,  175.3172942 ,
               -47.72876416, -226.37327686,  133.56786494,  -73.60743201,
                76.39514766,  -85.05313725,   34.6008544 ,   73.91773989,
                34.51007299,  -46.55912865,   14.67549864,  -97.09800107,
               -45.50963162,   88.6166623 ,   51.67087278, -229.01743694,
                23.20404538,   36.75471   ,  180.61408913,  -54.55405549,
                 8.76150565,  142.464305  ,   66.00475351,  247.22935104,
               -38.04510005,  159.42729079,  292.65710584,   93.53107115,
               -29.72737274,   31.42627031,   11.40385325,  -25.32078774])
```

In [7]:
```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 1708.40621428299
R-squared: 0.8933194542611739
```

Q4. What is a weak learner in Gradient Boosting?

In Gradient Boosting, a weak learner is a model that performs slightly better than random guessing. Typically, decision trees with small depths are used as weak learners in Gradient Boosting.

The reason for using weak learners is that they are simple and computationally efficient. They can be trained quickly and combined easily to form a powerful ensemble model. When combined, the weak learners can compensate for each other's weaknesses and produce a model that is much stronger than any individual weak learner.

The role of the weak learner in Gradient Boosting is to make incremental improvements to the model's predictive accuracy. The weak learner is trained on the errors of the previous learner, with the goal of reducing those errors by a small amount. By repeatedly adding weak learners, the model can gradually improve its accuracy on the training data.

Since the weak learners in Gradient Boosting are simple and have low variance, they are less likely to overfit the training data. This makes the model more robust and less prone to errors on new, unseen data.

Q5. What is the intuition behind the Gradient Boosting algorithm?

The intuition behind the Gradient Boosting algorithm is to build a powerful ensemble model by iteratively adding weak learners (decision trees) to the model, each one correcting the errors made by the previous learners. The algorithm achieves this by minimizing the loss function of the model, which measures the difference between the predicted values and the actual values of the target variable.

At each iteration, the algorithm trains a new decision tree on the errors of the previous iteration. The tree is trained to predict the residual errors of the previous model. The residuals represent the difference between the predicted and actual values of the target variable, and the new decision tree aims to reduce these residuals.

The new decision tree is then added to the ensemble, and its predictions are combined with the predictions of the previous models to obtain a better overall prediction. This process is repeated for a predefined number of iterations, with each iteration producing a new decision tree that further improves the overall model. The key idea behind Gradient Boosting is that each subsequent tree tries to correct the mistakes made by the previous tree, and the algorithm adjusts the weights of the samples in each iteration to focus on the samples that were not correctly predicted by the previous trees. By doing this, the algorithm can gradually improve the accuracy of the model and achieve better predictions on new, unseen data.

Q6. How does Gradient Boosting algorithm build an ensemble of weak learners?

Train the initial weak learner: The first weak learner is trained on the training data to predict the target variable. This weak learner can be any model that performs slightly better than random guessing, but decision trees with small depths are commonly used. Compute the residuals: The predicted values of the first weak learner are subtracted from the actual values of the target variable to compute the residuals. These residuals represent the difference between the predicted and actual values and are used as the target variable for the next weak learner. Train subsequent weak learners: The subsequent weak learners are trained on the residuals of the previous weak learner. The goal is to minimize the residuals of the previous learner by fitting a new weak learner. Add the weak learner to the ensemble: The new weak learner is added to the ensemble of models, and its predictions are combined with the predictions of the previous models to obtain a better overall prediction. Repeat steps 2-4 until a stopping criterion is met: The process of adding weak learners is repeated until the ensemble of models reaches a predefined number of models or until the model's performance on the validation data stops improving.

Q7. What are the steps involved in constructing the mathematical intuition of Gradient Boosting algorithm?

Train the initial weak learner: The first weak learner is trained on the training data to predict the target variable. This weak learner can be any model that performs slightly better than random guessing, but decision trees with small depths are commonly used. Compute the residuals: The predicted values of the first weak learner are subtracted from the actual values of the target variable to compute the residuals. These residuals represent the difference between the predicted and actual values and are used as the target variable for the next weak learner. Train subsequent weak learners: The subsequent weak learners are trained on the residuals of the previous weak learner. The goal is to minimize the residuals of the previous learner by fitting a new weak learner. Add the weak learner to the ensemble: The new weak learner is added to the ensemble of models, and its predictions are combined with the predictions of the previous models to obtain a better overall prediction. Repeat steps 2-4 until a stopping criterion is met: The process of adding weak learners is repeated until the ensemble of models reaches a predefined number of models or until the model's performance on the validation data stops improving.

In [ ]: