28-Gradient_Boosting_Regressor

October 20, 2024

1 Gradient Boosting Regressor

A **Gradient Boosting Regressor** is a machine learning algorithm that builds an ensemble of decision trees sequentially to minimize the loss function (or error) for regression tasks. Unlike Random Forest, which builds trees independently, Gradient Boosting builds trees sequentially, where each new tree aims to correct the errors made by the previous trees. It uses the gradient descent technique to optimize the model by reducing the difference between the predicted and actual values.

It combines the predictions of many "weak learners" (typically shallow decision trees) to create a more accurate final prediction. The key idea is to reduce bias and variance by focusing on improving areas where the model previously performed poorly.

1.0.1 Why is Gradient Boosting Regressor Important?

- **Accurate Predictions**: Gradient Boosting often produces very accurate models that outperform simpler models like linear regression, and even Random Forest, in certain cases.
- **Focuses on Hard-to-Predict Instances**: By iteratively correcting errors, Gradient Boosting effectively captures complex patterns in the data that may be missed by other models.
- **Balances Bias-Variance Tradeoff**: Gradient Boosting reduces both bias (error due to underfitting) and variance (error due to overfitting), providing a powerful model in various situations.

1.0.2 How does Gradient Boosting Regressor work?

The core idea behind Gradient Boosting is to add new models that improve the residual errors (i.e., the difference between predicted and true values) of previous models. Here's the step-by-step process:

1. Initial Model: Start with an initial model, typically predicting the mean of the target variable.

2. Calculate Residuals*: For each data point, compute the residual (the difference between the true value and the model's prediction).

3. Fit a Weak Learner: A decision tree is fitted to these residuals. This new tree learns to predict the errors (or residuals) made by the previous model.

4. Update the Model: The new predictions are added to the previous predictions. This step is scaled by a learning rate to control the contribution of the new tree.

5. Repeat: This process is repeated iteratively, each time fitting a new tree to the residuals and updating the model.

6. Final Prediction: After a certain number of iterations (trees), the final model is the sum of the predictions from all trees.

The key difference between Gradient Boosting and Random Forest is that Gradient Boosting builds trees sequentially, with each new tree correcting the errors made by the previous trees, whereas Random Forest builds trees independently in parallel.

1.0.3 When should you use Gradient Boosting Regressor?

- **When high accuracy is needed**: Gradient Boosting is particularly powerful when you need highly accurate predictions and are willing to trade off some interpretability.
- **Imbalanced or Noisy Data**: Gradient Boosting handles imbalanced and noisy data well, making it suitable for real-world applications where clean, well-balanced datasets are rare.
- **When dealing with complex relationships**: If the data exhibits complex relationships between features and target variables, Gradient Boosting can model these effectively.
- **When feature importance is important**: Like Random Forest, Gradient Boosting also provides feature importance, which can help in understanding which features contribute the most to the predictions.

1.0.4 Who uses Gradient Boosting Regressor?

- **Data Scientists and Machine Learning Engineers**: Gradient Boosting is widely used in industries like finance, healthcare, and marketing for predictive modeling tasks where high accuracy is crucial.
- **Kaggle Competitors**: Gradient Boosting models, especially variants like XGBoost, LightGBM, and CatBoost, are commonly used in data science competitions because of their ability to achieve state-of-the-art performance.
- **Researchers**: In research fields like genetics or climate modeling, Gradient Boosting is used for regression tasks that require capturing non-linear patterns and interactions among features.

1.0.5 Key Points to Remember:

- **Sequential Trees**: Gradient Boosting builds trees one at a time, with each tree focusing on correcting the residual errors of the previous trees.
- **Learning Rate**: The learning rate controls how much the new tree's predictions influence the overall model. A lower learning rate requires more trees to be effective but helps prevent overfitting.
- **Combining Weak Learners**: It combines many "weak learners" (typically small decision trees) to create a strong predictive model.
- **Loss Function**: Gradient Boosting minimizes a loss function (e.g., Mean Squared Error for regression) by performing gradient descent in function space.

1.0.6 Advantages of Gradient Boosting Regressor:

1. High Predictive Accuracy: Gradient Boosting tends to produce highly accurate models that often outperform other models on many datasets.

2. Flexibility: You can specify different loss functions (e.g., least squares for regression) and customize the model's hyperparameters.

3. Handles Missing Data: Gradient Boosting can handle missing data quite well and does not require complex imputation strategies.

4. Feature Importance: It provides feature importance, which can be useful for understanding which features have the most predictive power.

5. Robust to Outliers: It focuses on reducing residual errors, making it more robust to outliers in some cases.

1.0.7 Limitations of Gradient Boosting Regressor:

- **Longer Training Time**: Since Gradient Boosting builds trees sequentially, training can take significantly longer than other algorithms like Random Forest, especially on large datasets.
- **Sensitive to Hyperparameters**: Gradient Boosting can be sensitive to the choice of hyperparameters, such as the number of trees, learning rate, and tree depth. It often requires careful tuning to perform optimally.
- **Can Overfit**: Although Gradient Boosting generally performs well, it can overfit the training data if the number of trees is too high or if the trees are too deep.
- **Not Interpretable**: Like other ensemble methods, Gradient Boosting produces complex models that are difficult to interpret compared to simpler models like linear regression.

[1]:
```
import pandas as pd

from sklearn import datasets
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, root_mean_squared_error,␣
 ↪r2_score

diabetes = datasets.load_diabetes()
diabetes.data
```

[1]:
```
array([[ 0.03807591,  0.05068012,  0.06169621, …, -0.00259226,
         0.01990749, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, …, -0.03949338,
        -0.06833155, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, …, -0.00259226,
         0.00286131, -0.02593034],
       …,
       [ 0.04170844,  0.05068012, -0.01590626, …, -0.01107952,
        -0.04688253,  0.01549073],
```

```
        [-0.04547248, -0.04464164,  0.03906215, …,  0.02655962,
          0.04452873, -0.02593034],
        [-0.04547248, -0.04464164, -0.0730303 , …, -0.03949338,
         -0.00422151,  0.00306441]])
```

[2]:
```
diabetes.target
```

[2]:
```
array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
        69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
        68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
        87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
       259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
       128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
       150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
       200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
        42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
        83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
       104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
       173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
       107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
        60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
       197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
        59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
       237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
       143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
       142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
        77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
        78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
       154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
        71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  91.,
       150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 116.,
       145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  66.,
        94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 233.,
        60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  89.,
        31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 172.,
       114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 109.,
       191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 135.,
       244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 216.,
       263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  71.,
        77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 321.,
        58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 168.,
       140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  69.,
       219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 258.,
        43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  72.,
       140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  55.,
        84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 310.,
        94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 132.,
```

```
        220.,  57.])
```

[3]:
```
X = diabetes.data
y = diabetes.target

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
 ↪random_state=17)

gbr = GradientBoostingRegressor()
gbr.fit(X_train, y_train)
```

[3]:
```
GradientBoostingRegressor()
```

[4]:
```
y_pred = gbr.predict(X_test)
y_pred
```

[4]:
```
array([192.51225586, 200.5750764 ,  82.78231894, 101.58510513,
       125.14414245,  81.39853646,  93.84628461, 102.45928951,
       121.90945671,  74.25891122, 255.91425718, 175.42143255,
        93.81164289,  99.4616262 , 217.43499676, 217.43197684,
       190.62007333, 172.98697169, 197.2837651 , 175.80982432,
       182.09395792, 269.70763529, 196.07382183,  82.25148771,
       114.47110984, 185.87258222, 109.04630496, 165.3091288 ,
        84.08079017, 241.11508989,  89.77664274,  69.54907641,
       108.03496427, 288.56040447, 170.41296018, 164.48100696,
        82.93345568, 179.53609042,  91.32177689, 245.58054393,
       112.63272196,  90.77107175, 179.75704585, 306.48853493,
       154.11016096,  91.65230129, 200.49974318, 107.20195482,
       190.62028854, 167.72815926, 191.98143713, 186.27531144,
       200.12776418, 156.00397936, 180.02678304, 138.30178657,
        96.52611334, 300.64236743, 138.20868823, 162.7712336 ,
       176.10010144, 231.10141163,  91.57716704, 143.7471825 ,
        87.83391334,  91.52052802, 216.22481281,  68.19976982,
       167.75688514, 157.00162733, 103.12604888, 283.27021778,
       267.45306696, 196.29244184,  96.48618333, 124.81762663,
       199.10839318, 149.25733184, 145.53020157, 178.10088311,
       258.17118587,  92.95097597, 149.38905097,  71.33146438,
       272.96959869, 311.01641671,  95.577347  , 104.14250789,
        83.75796396])
```

[5]:
```
mean_absolute_error(y_test, y_pred)
```

[5]:
```
48.792591465108316
```

[6]:
```
root_mean_squared_error(y_test, y_pred)
```

[6]:
```
60.90137361126108
```

[7]:
```
r2_score(y_test, y_pred)
```

[7]:
```
0.37191596197265364
```

[8]:
```
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3,4,5],
    'min_samples_split': [2,3,4],
    'min_samples_leaf': [1,2,3]
}

gbr_cv = GridSearchCV(gbr, param_grid, cv=5,␣
 ↪n_jobs=-1,scoring='neg_root_mean_squared_error')
gbr_cv.fit(X_test, y_test)
```

[8]:
```
GridSearchCV(cv=5, estimator=GradientBoostingRegressor(), n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.1, 0.2],
                         'max_depth': [3, 4, 5], 'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [100, 200, 300]},
             scoring='neg_root_mean_squared_error')
```

[9]:
```
y_pred_gbr_cv = gbr_cv.predict(X_test)
y_pred_gbr_cv
```

[9]:
```
array([167.54547384, 196.45689046, 131.99856228, 128.81980861,
       155.38736053, 104.78343157, 104.46878826, 103.86864233,
       172.16237464,  92.877811  , 226.28290431, 151.96615636,
        99.36099017, 184.74380459, 209.7874883 , 169.38985178,
       240.03918274, 210.69910838, 247.98095965, 113.01953415,
       110.64514541, 232.66480304, 109.31434018,  91.94434266,
        96.51531206, 215.20968459, 105.09234396, 138.79638199,
        82.32851947, 215.84795964, 119.4373777 , 116.85111785,
       109.52830044, 199.97027303, 126.65963299, 231.83342139,
        79.28364229, 253.17948242, 107.99860644, 248.12009523,
        95.73091409, 134.91997317, 165.55760765, 241.81722869,
       107.02535237,  78.75843641, 211.98096742,  99.36099017,
       137.05366889, 163.12544504, 164.34451529, 178.11208294,
       182.44224851, 114.1775484 , 201.79323332, 104.77956892,
       115.67104533, 233.66677325, 163.98975366, 221.99947933,
       198.83299795, 172.19115755,  90.89484407, 123.40323468,
       131.31453483, 191.68730783, 167.43521912, 105.88265868,
       187.91161719, 150.97652913, 119.96678341, 241.33644356,
       257.04655152, 164.34451529,  99.89637165, 143.87516951,
       101.74985603, 243.13993649, 152.49661722, 231.19421556,
       233.42395538, 120.44390831, 140.12835633,  81.63035609,
```

```
       236.03992088, 239.51517181, 117.57491008, 104.18911766,
       104.31100263])
```

[10]:
```
mean_absolute_error(y_test, y_pred_gbr_cv)
```

[10]:
```
27.560044315482546
```

[11]:
```
root_mean_squared_error(y_test, y_pred_gbr_cv)
```

[11]:
```
32.99791585313554
```

[12]:
```
r2_score(y_test, y_pred_gbr_cv)
```

[12]:
```
0.8156103237854524
```

[13]:
```
gbr_cv.best_params_
```

[13]:
```
{'learning_rate': 0.01,
 'max_depth': 3,
 'min_samples_leaf': 3,
 'min_samples_split': 3,
 'n_estimators': 200}
```