Final Project: Gradient Boosting

Marco Ren

Section 1: Introduction

In many machine learning tasks, a common goal is to build a predictive model from a given dataset and to ensure that the predictive model generalizes well to new datasets. However, individual models, such as OLS regressions and decision trees, often fail to capture the complexities in the data and, therefore, perform poorly. In this case, ensemble methods such as gradient boosting are better solutions for improving the model. In this paper, we will focus on the basic idea of gradient boosting for regression, including simple implementation and regularization methods.

Gradient boosting is a powerful machine learning technique that has been proven to be useful in multiple scenarios. In practice, it takes multiple weak models, regression trees in this case, and combines them to create a robust predictive model. The key idea is to add weak models sequentially to correct errors in the combined model. Each new model is trained utilizing the concept of gradient descent in function space to minimize a loss function, which measures the error of the model on the training data. More specifically, at each iteration, the algorithm calculates the gradient of the loss function with respect to the prediction of the current model and uses it to create a new weak model that focuses on areas where the current model has performed poorly.

This paper is based on the article "Gradient boosting machines, a tutorial" by Natekin A. and Knoll A. [1], which discusses the basic implementation of gradient boosting and the possible regularization methods. To get a thorough understanding of gradient boosting, we have to first take a look at the concept of regression trees.

Section 2: Regression Tree

Regression Tree is a simple predictive model that partitions the predictor space into a number of small regions, with each region corresponding to a different value of the target variable. The tree is built by iteratively splitting the data based on the values of the predictors until a certain stopping point is reached. The leaf nodes at the bottom of the tree contain the predicted values of the target variable in each region, usually averaged over the values of the target variable in the region. Take the following figure as an example:

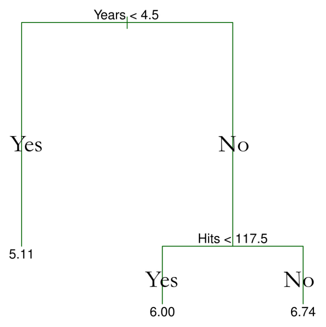


Figure 1

The figure above is adapted from the PDF "Tree based methods: Regression trees" by Rebecca, S. C. [3]. In this example, we want to predict the log salary of a baseball player based on the predictors $Year$ (the number of years in the major leagues) and $Hits$ (the number of hits made in the previous year). Each leaf node in the regression tree contains the predicted value of log salary that falls into the region defined by the specific combinations of $Year$ and $Hits$ splits made by the tree.

Unlike OLS regression, which can only capture linear relationships, the algorithm for regression trees allows for nonlinear relationships to be captured. However, using only one single regression tree tends to underfit the data. In this case, we need gradient boosting to improve the model.

Section 3: Gradient Boosting Methodology

3.1: Loss Function

Gradient boosting focuses on estimating a function $f(x)$ that maps predictors $x \in \mathbb{R}^d$ to the target variable y while minimizing the predetermined loss function $\ell(y, f(x))$, which measures how well the estimated function predicts the true value of the target variable y.

Note that the loss function is predetermined. Since this paper only discusses regression problems, we can use either the absolute L1 loss function or the squared error L2 loss function, both of which are commonly used for continuous target variable y. The absolute L1 loss function takes the form of:

$$\ell(y, f(x))_{L_1} = |y - f(x)| \tag{1}$$

It is more robust to outliers and tends to lead to sparsity in the parameters of the model compared to the squared error L2 loss function. The squared error L2 loss function takes the form of:

$$\ell(y, f(x))_{L_2} = \frac{1}{2}(y - f(x))^2 \tag{2}$$

Compared to the absolute L1 loss function, the squared error L2 loss function penalizes large errors more heavily, leading to smooth and stable optimization, which makes it effective when outliers are not a significant concern.

3.2: Gradient Descent in Parameter Space

Given the loss function, the goal of optimization is to find the function $\hat{f}(x)$ that can minimize the expected loss:

$$\hat{f}(x) = \arg\min_f \mathbb{E}_{x,y}[\ell(y, f(x))] \tag{3}$$

In supervised learning, the given dataset typically contains a finite set of n observations, which can be written as $\{(x_i, y_i)\}_{i=1}^n$. Then, the minimization of the expected loss can be expressed as:

$$\hat{f}(x) = \arg\min_f \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(x_i)) \tag{4}$$

Since $\hat{f}(x)$ is typically unknown, the functional space is restricted to $f(x, \theta)$, where θ represents the parameters. Then, the optimization problem can be rewritten as:

$$\hat{\theta} = \arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(x_i, \theta)) \tag{5}$$

In addition, the objective function $J(\theta)$, which represents the total loss, can be written as:

$$J(\theta) = \sum_{i=1}^{n} \ell(y_i, f(x_i, \hat{\theta})) \tag{6}$$

To solve the optimization problem, we can apply the gradient descent method with the initial parameter estimates $\hat{\theta}_0$:

$$\hat{\theta}_{t+1} = \hat{\theta}_t - \rho \nabla J(\theta_t) \tag{7}$$

Here, $\rho > 0$ represents the step size and $J(\theta)$ represents the gradient of the objective function $J(\theta)$ with respect to θ. Note that this iterative approach does not require explicit assumptions about the form of $f(x)$.

3.3: Gradient Descent in Function Space

After building the foundation of gradient descent in parameter space, we can move on to discuss gradient descent in function space, where the algorithm of gradient boosting actually performs optimization. In traditional machine learning algorithms such as OLS regressions, the function is typically given by a fixed parametric model, and optimization is implemented by adjusting the parameters. However, gradient boosting performs optimization by iteratively improving the function it is learning, where the function $f(x)$ is represented as the sum of the previous models and a series of weak learners $h(x, \theta)$ that correct the errors made by the previous models. More specifically, the iteration approach can be written as:

$$\hat{f}_t(x) = \hat{f}_{t-1}(x) + \rho_t h(x, \theta_t) \tag{8}$$

Here, ρ_t is the step size and $h(x, \theta_t)$ is the weak learner trained at the t-th iteration to correct the errors made by the previous models. Note that in this case, the weak learner $h(x, \theta_t)$ is the approximation of the negative gradient of the loss function, whose components can be computed as:

$$g_t(x_i) = -\frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)}\Big|_{f(x)=\hat{f}_{t-1}(x)} \tag{9}$$

Here, $g_t(x_i)$ is called the pseudo-residuals, which represents the difference between the model prediction $f_{t-1}(x_i)$ and the true values of the target variable y_i (how much the model prediction needs to change to reduce the loss). The goal is to find $h(x, \theta_t)$ so that the next model prediction is as close as possible to the true values of the target variable y_i. Therefore, we can find $h(x, \theta_t)$ that minimize the squared error between the pseudo-residuals $g_t(x_i)$ and the weak learner $h(x_i, \theta)$:

$$\hat{h}_t(x, \theta_t) = \arg\min_h \sum_{i=1}^{n} \left(-g_t(x_i) + h(x_i, \theta)\right)^2 \tag{10}$$

Similarly, since $\hat{h}_t(x, \theta_t)$ is unknown, we can rewrite the equation to:

$$\hat{\theta}_t = \arg\min_\theta \sum_{i=1}^{n} \left(-g_t(x_i) + h(x_i, \theta)\right)^2 \tag{11}$$

Once the weak learner is fitted, the next step is to find the best gradient descent step size ρ_t by minimizing the loss function after we determine the weak learner:

$$\rho_t = \arg\min_\rho \sum_{i=1}^{n} \ell\left(y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)\right) \tag{12}$$

Therefore, the iteration approach can be written as:

$$\hat{f}_t(x) = \hat{f}_{t-1}(x) + \rho_t h(x, \theta_t) \tag{13}$$

By iteratively adding new weak learners and refining the function $f(x)$ in the function space, gradient boosting can capture more complex data patterns without having to globally optimize all parameters at once. This is particularly important for problems involving large datasets or high-dimensional feature spaces, where traditional optimization methods may be computationally expensive or difficult to apply.

3.4: Algorithm Summary

To summarize the methodology of gradient boosting implementation, the pseudo-code for the basic algorithm is provided below:

Algorithm 1 Friedman's Gradient Boost algorithm

```
Inputs:
  • input data $(x, y)_{i=1}^{n}$
  • number of iterations $M$
  • choice of the loss-function $\ell(y, f)$
  • choice of the base-learner model $h(x, \theta)$
Algorithm:
  1. Initialize $\hat{f}_0$ with a constant
  2. for $t = 1$ to $M$ do
     (a) Compute the negative gradient $g_t(x)$
     (b) Fit a new base-learner function $h(x, \theta_t)$:
           $\hat{\theta}_t = \arg\min_{\theta} \sum_{i=1}^{n} \left(-g_t(x_i) + h(x_i, \theta)\right)^2$
     (c) Find the best gradient descent step-size $\rho_t$:
           $\rho_t = \arg\min_{\rho} \sum_{i=1}^{n} \ell\left(y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)\right)$
     (d) Update the function estimate:
           $\hat{f}_t(x) = \hat{f}_{t-1}(x) + \rho_t h(x, \theta_t)$
  3. end for
```

Section 4: Gradient Boosting Regularization

Recall that the purpose of building a predictive model is to ensure that it generalizes well to new datasets. In this case, while gradient boosting can often capture complex patterns in the data, they can sometimes overfit the data, especially when applied to noisy datasets or small sample sizes. In this section, we will introduce different regularization methods to reduce the risk of overfitting.

4.1: Subsampling

Subsampling is an effective regularization technique in gradient boosting. The use of subsampling refers to randomly selecting a fraction of the training data in each iteration to train each base learner. By introducing randomness into the learning process, subsampling helps prevent the model from fitting too closely to the training data, which can lead to overfitting and poor performance.

One key advantage of subsampling is that it naturally reduces computational complexity. By training smaller subsamples of the data in each iteration, the implementation of gradient boosting requires less time and memory storage and is, therefore, well-suited for large datasets. However, the fraction of data selected at each iteration needs to be determined manually. While a smaller subsample results in greater randomness and better model generalization, it comes at the potential cost of increasing bias, which means that the model may underfit the data if too small a fraction of the data is selected in each iteration. In this case, the model may fail to capture important patterns in the data, leading to decreased accuracy.

4.2: Shrinkage

Shrinkage, also can be referred to as the learning rate, is another important regularization technique in gradient boosting. It works by shrinking the contribution of each weak learner in the overall model. More specifically, the weak learner term in each iteration is multiplied by a fixed shrinkage parameter λ:

$$\hat{f}_t(x) = \hat{f}_{t-1}(x) + \lambda \rho_t h(x, \theta_t) \tag{14}$$

Note that the shrinkage parameter typically takes a value between 0 and 1 and is fixed throughout the learning process. In the context of regularization, one can reduce the learning rate λ. By slowing down the learning process, shrinkage ensures that the model improves gradually, reducing the risk of overfitting.

One of the key advantages of shrinkage is that it strikes a balance between model generalization and model complexity. When the value of the learning rate decreases, the algorithm will use a larger number of iterations to minimize the loss function, allowing the model to be built by small incremental improvements. These gradual improvements help the model to focus more on meaningful patterns in the data rather than noise, which reduces the risk of overfitting. However, shrinkage also introduces computational trade-offs. A smaller value of λ requires a larger number of iterations to reach convergence, which will take more time and memory storage in the learning process.

4.3: Early Stopping

In the previous section, a disadvantage of shrinkage mentioned is that it significantly raises the computational cost due to the increased number of iterations. One regularization technique in gradient boosting that can help mitigate this issue is early stopping. Early stopping works by monitoring the performance of the model during the learning process and stopping the iterations once the model stops improving. By terminating the learning process early, early stopping prevents the model from becoming too complex and fitting noises in the training data, which can lead to overfitting.

One key advantage of early stopping is its ability to dynamically adjust the number of iterations, eliminating the need to manually predetermine the number of iterations. In addition, the early stopping approach complements other regularization methods, such as shrinkage and subsampling, as it ensures that the model does not continue to improve itself with negligible improvement, ultimately leading to overfitting.

Section 5: Experiment

To demonstrate the ability of gradient boosting to capture complex patterns in data compared to simpler models such as OLS regressions and individual regression trees, as well as to explore the role of different regularization methods in improving the performance of gradient boosting, an experiment is designed using the "California housing prices" dataset by Nugent C. [2]. The dataset contains variables including:

1. *longitude*: A measure of how far west a house is; a higher value is farther west
2. *latitude*: A measure of how far north a house is; a higher value is farther north
3. *housingMedianAge*: Median age of a house within a block; a lower number is a newer building
4. *totalRooms*: Total number of rooms within a block
5. *totalBedrooms*: Total number of bedrooms within a block
6. *population*: Total number of people residing within a block

7. $households$: Total number of households, a group of people residing within a home unit, for a block

8. $medianIncome$: Median income for households within a block of houses (measured in tens of thousands of US Dollars)

9. $oceanProximity$: Location of the house w.r.t ocean/sea

10. $medianHouseValue$: Median house value for households within a block (measured in US Dollars), which is the target variable we aim to predict using the above predictors.

In the experiment, a gradient boosting predictive model for regression will be developed to predict the target variable $medianHouseValue$ using the predictors. Note that the predictor $oceanProximity$ is categorical with five categories: $NEARBAY$, $< 1HOCEAN\ INLAND$, $NEAROCEAN$, and $ISLAND$. We will create a dummy variable and assign each category a number from 1 to 5.

5.1: Data Exploration

We will start by exploring the dataset. First, we will plot the relationship between the numerical predictors and the target variable.

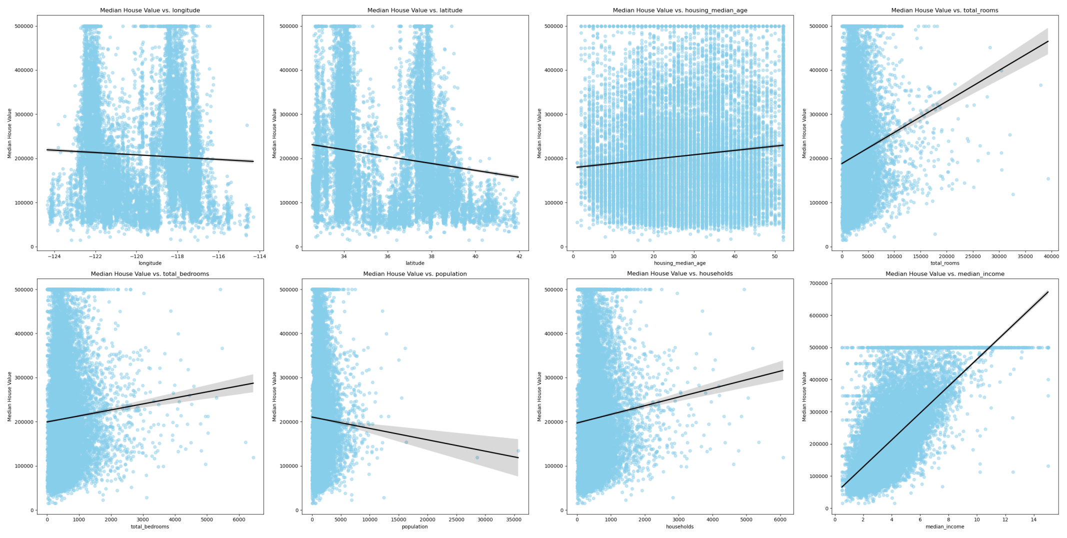


Figure 2

According to Figure 2, there is a strong positive correlation between the predictors $totalRooms$ and $medianIncome$ and the target variable $medianHouseValue$. In contrast, the predictors $housingMedianAge$, $totalBedrooms$, and $households$ and the target variable $medianHouseValue$ are weakly positively correlated and the predictors $longitude$, $latitude$, and $households$ and the target variable $population$ are weakly negatively correlated.

Then, We will proceed to plot the relationship between the categorical predictor $oceanProximity$ and the target variable.

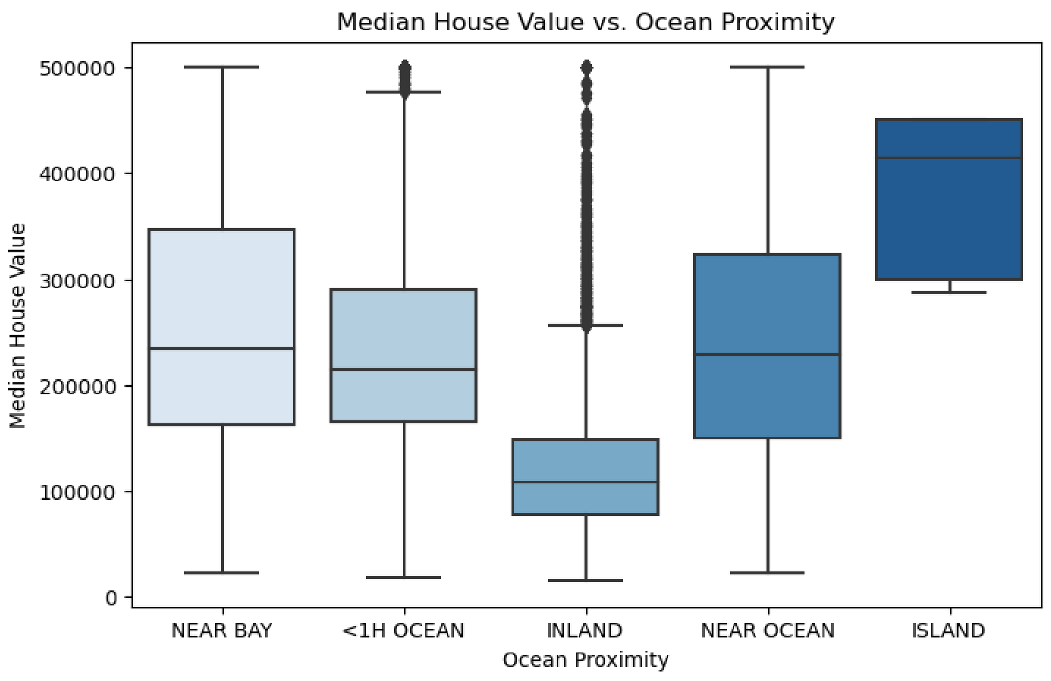
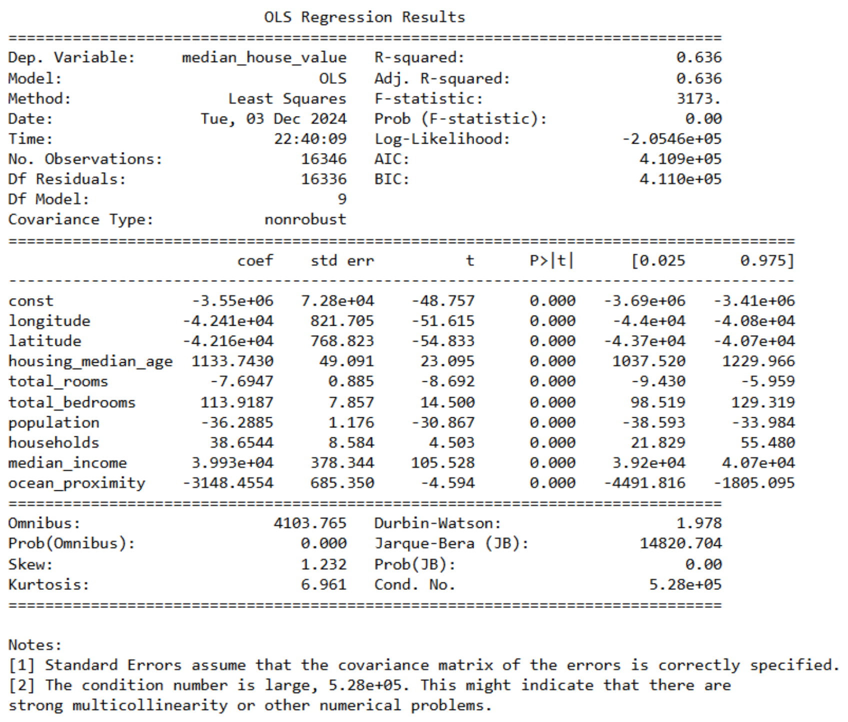


Figure 3

According to Figure 3, inland houses generally have the smallest median house value and island houses generally have the largest median average house value.

5.2: Predictive Model

Since there may exist multicollinearity, which means there may be correlations between predictors, we need a predictive model that is more robust. We will start by implementing the simple OLS regression. The following figure demonstrates the results of the OLS regression:



```
                            OLS Regression Results
==============================================================================
Dep. Variable:     median_house_value   R-squared:                       0.636
Model:                            OLS   Adj. R-squared:                  0.636
Method:                 Least Squares   F-statistic:                     3173.
Date:                Tue, 03 Dec 2024   Prob (F-statistic):               0.00
Time:                        22:40:09   Log-Likelihood:            -2.0546e+05
No. Observations:               16346   AIC:                         4.109e+05
Df Residuals:                   16336   BIC:                         4.110e+05
Df Model:                           9
Covariance Type:            nonrobust
======================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------
const               -3.55e+06   7.28e+04    -48.757      0.000   -3.69e+06   -3.41e+06
longitude          -4.241e+04    821.705    -51.615      0.000    -4.4e+04   -4.08e+04
latitude           -4.216e+04    768.823    -54.833      0.000   -4.37e+04   -4.07e+04
housing_median_age  1133.7430     49.091     23.095      0.000    1037.520    1229.966
total_rooms           -7.6947      0.885     -8.692      0.000      -9.430      -5.959
total_bedrooms       113.9187      7.857     14.500      0.000      98.519     129.319
population           -36.2885      1.176    -30.867      0.000     -38.593     -33.984
households            38.6544      8.584      4.503      0.000      21.829      55.480
median_income       3.993e+04    378.344    105.528      0.000    3.92e+04    4.07e+04
ocean_proximity    -3148.4554    685.350     -4.594      0.000   -4491.816   -1805.095
==============================================================================
Omnibus:                     4103.765   Durbin-Watson:                   1.978
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            14820.704
Skew:                           1.232   Prob(JB):                         0.00
Kurtosis:                       6.961   Cond. No.                     5.28e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.28e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Figure 4

According to the results of the OLS regression, all predictor variables are significant in predicting the target variable because the p-values of all predictors are very small. However, the test root mean squared error computed using this model is about 69274.9517, which is very large. This is because the OLS regression can only capture linear relationships and we need a model that can capture non-linear ones. Therefore we need to proceed to construct a regression tree:

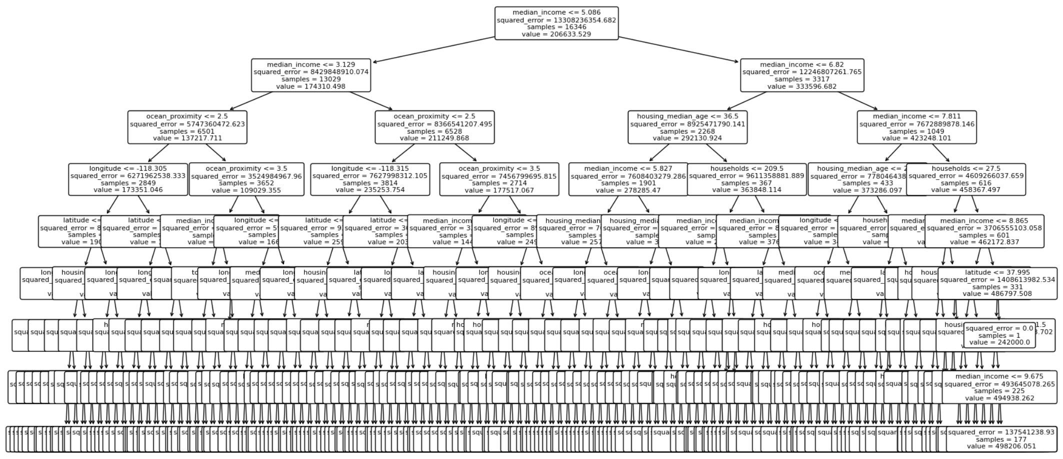


Figure 5

The test root mean squared error computed using this model is approximately 62437.8555, which is slightly smaller than the OLS regression since the regression tree can capture nonlinear relationships. Note that we limit the maximum depth of the tree to 8 in this case to make the tree look clearer and to prevent overfitting. However, we can still improve the model to capture more complex patterns in the data by implementing gradient boosting:

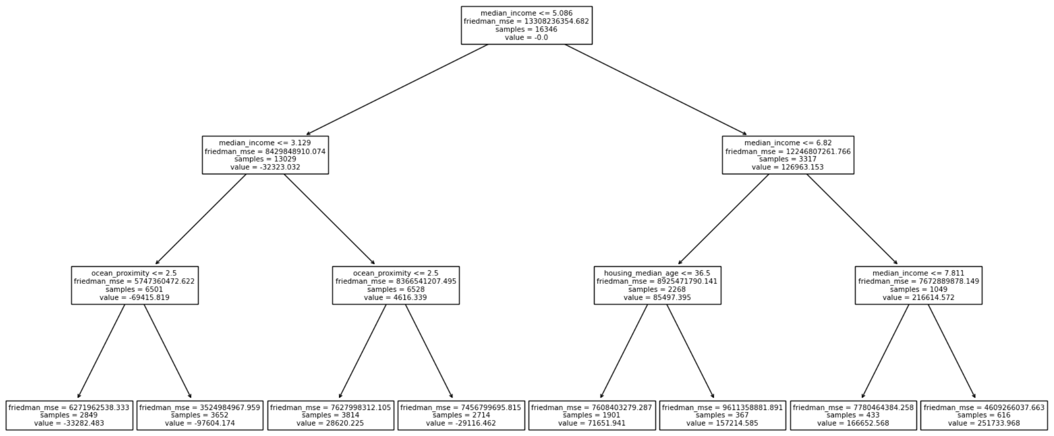


Figure 6

The test root mean squared error computed using gradient boosting is approximately 50552.0539, which is much smaller than the regression tree since gradient boosting captures more complex patterns in the data. Note that we limit the maximum depth of the trees to 3 in this case to make the trees look clearer and to prevent overfitting. However, there are still ways to improve the model. Since the nature of gradient boosting inherently increases the risk of overfitting, we can improve the model by implementing different regularization techniques.

5.3: Regularization

5.3.1: Subsampling

We will first implement subsampling. Here, we randomly select 97.5 percent of the training data in each iteration to train each base learner and check if we can obtain a smaller test root mean squared error. Note that we do not want to set the subsample size too small due to underfitting.

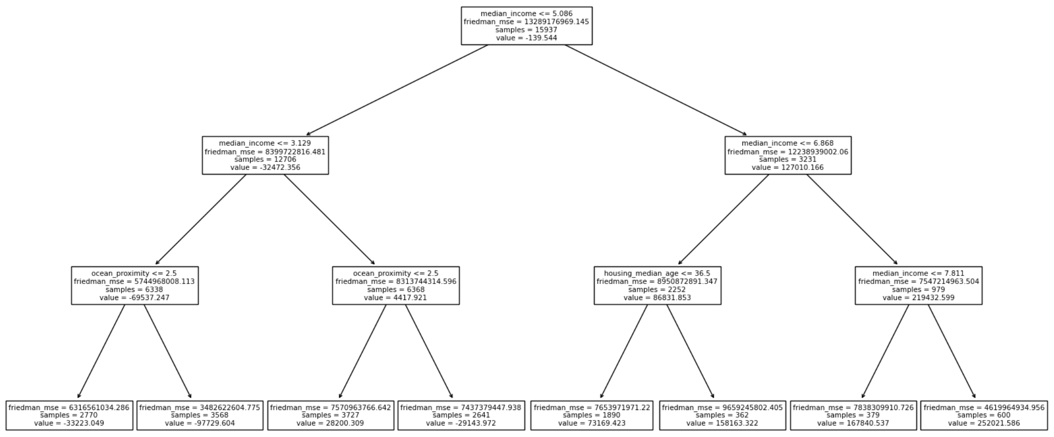


Figure 7

Note that after implementing subsampling, the test root mean squared error reduces from 50552.0539 to 50110.9774. The small reduction indicates that the training data may contain repetitive or highly correlated information across many different features. Therefore, the model is still able to learn most of the correlation patterns even if only a subset of the data is used in each iteration.

5.3.2: Shrinkage

We will then implement shrinkage, which can also be referred to as the learning rate. Here, we slow down the learning process by decreasing the value of the learning rate from 0.5 to 0.1 and check if we can obtain a smaller test root mean squared error.

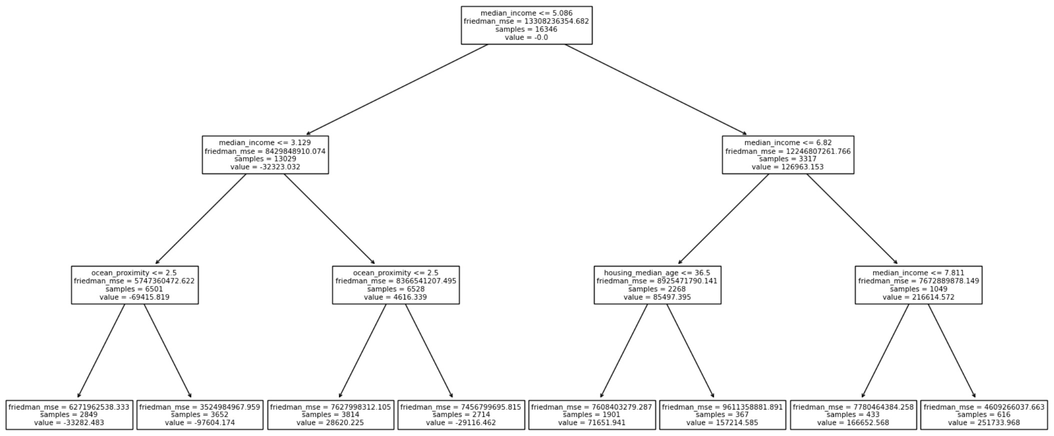


Figure 8

After implementing shrinkage, the test root mean squared error reduces from 50552.0539 to 47408.8197, which is a significant improvement. This large reduction is due to the slower learning process that forces the model to improve more gradually to prevent overfitting.

5.3.3: Early Stopping

Finally, we can implement early stopping. Here, we monitor the performance of the model during the learning process and terminate the iterations once the improvement of the mean squared error in the model becomes less than $1e-05$ in 300 consecutive iterations.

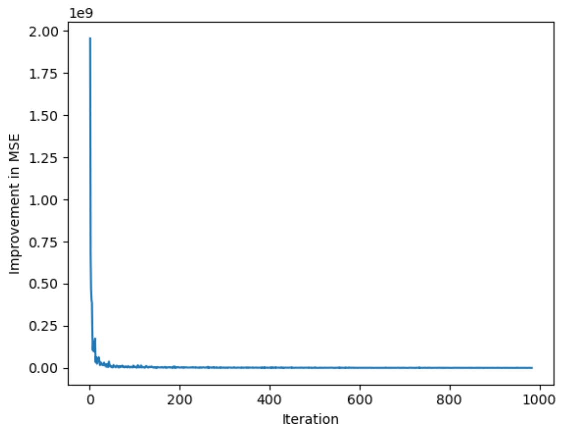


Figure 9

According to Figure 9, the improvement in the mean squared error decreases significantly as the number of iterations increases. The algorithm eventually terminates at about 1000 iterations.

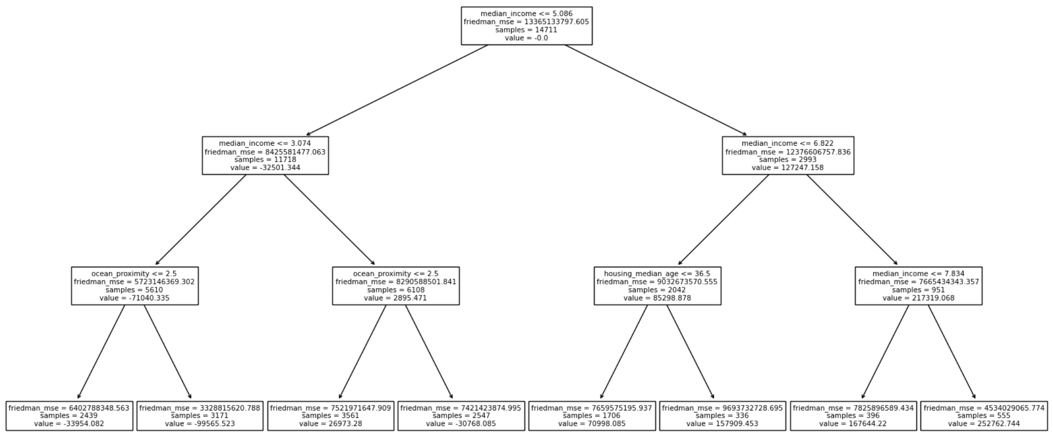


Figure 10

After implementing early stopping, the test root mean squared error reduces from 50552.0539 to 49663.1858. The reduction indicates that early stopping is able to determine an optimal termination point to terminate the learning process before the model begins to fit noises in the training data, thus preventing overfitting and allowing the model to better generalize to new datasets.

Section 6: Conclusion

In this paper, we introduce the basic idea of gradient boosting for regression problems, a powerful machine learning technique that can be used to iteratively improve the predictive accuracy of a model by combining weak learners, in this paper, regression trees. This paper includes the methodology of gradient boosting, as well as possible regularization methods for gradient boosting, such as subsampling, shrinkage, and early stopping, which can further improve the model by reducing the risk of overfitting and enhancing the ability to generalize to new datasets. These discussions provide a solid foundation for understanding gradient boosting and its practical applications, allowing this technique to be used more efficiently and reliably to solve complex regression tasks.

References

[1] A. Natekin and A. Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[2] C. Nugent. California housing prices [dataset]. Kaggle, November 2017. Retrieved from `https://www.kaggle.com/datasets/camnugent/california-housing-prices`.

[3] S. C. Rebecca. Tree based methods: Regression trees. Duke University, 2017.