Gradient Boosting

SPR Project, IIT Dharwad

S V Praveen, 170010025

Mandeep Bawa, 170030038

Ganesh Samarth, 170020030

Abstract

Gradient Boosting is a popular technique to iteratively combine weak learners to form a strong one. The report introduces the popular concept of gradient boosting for regression and classification. It then details the algorithms for each and demonstrates its performance on two famous datasets - Ames House Prices and Titanic, in comparison with other popular models and boosting techniques. The improvements that can be made to improve the gradient boosting model are discussed. Finally, we enlist the advantages and disadvantages of gradient boosting and also mentions the futility in using linear models as weak learners.

1. Introduction

Boosting is a family of machine learning algorithms that try to convert weak learners to strong learners. The underlying principle behind boosting was that any weak learner could be iteratively improved and transformed into a strong learner. There are many boosting algorithms - AdaBoost, Gradient Boosting, XGBoost. We mainly explore Gradient Boosting in this report but also mention and compare this method with other boosting algorithms and machine learning techniques in general.

2. Origin of Boosting

The idea of boosting was conceived while designing a transformation to convert weak learners to strong learners.

- ***Weak learner:*** *or a weak hypothesis is defined as a model whose performance is at least slightly better than random chance.*
- ***Strong learner:*** *or a strong hypothesis is a model which has been trained to perform nearly perfect classification.*

The roots of Boosting lie in a theoretical machine learning framework called the “PAC” learning model, Valiant [8,9]. Kearns and Valiant [10] were the first to pose the question of whether a “weak” learning algorithm which performs just slightly better than random guessing in the PAC model can be “boosted” into an arbitrarily accurate “strong” learning algorithm. Schapire [11] came up with the first provable polynomial-time boosting algorithm in 1989. A year later, Freund [12] developed a much more efficient boosting algorithm which, although optimal in a certain sense, nevertheless suffered from certain practical drawbacks. The first experiments with these early boosting algorithms were carried out by Drucker, Schapire and Simard [13] on an OCR task.

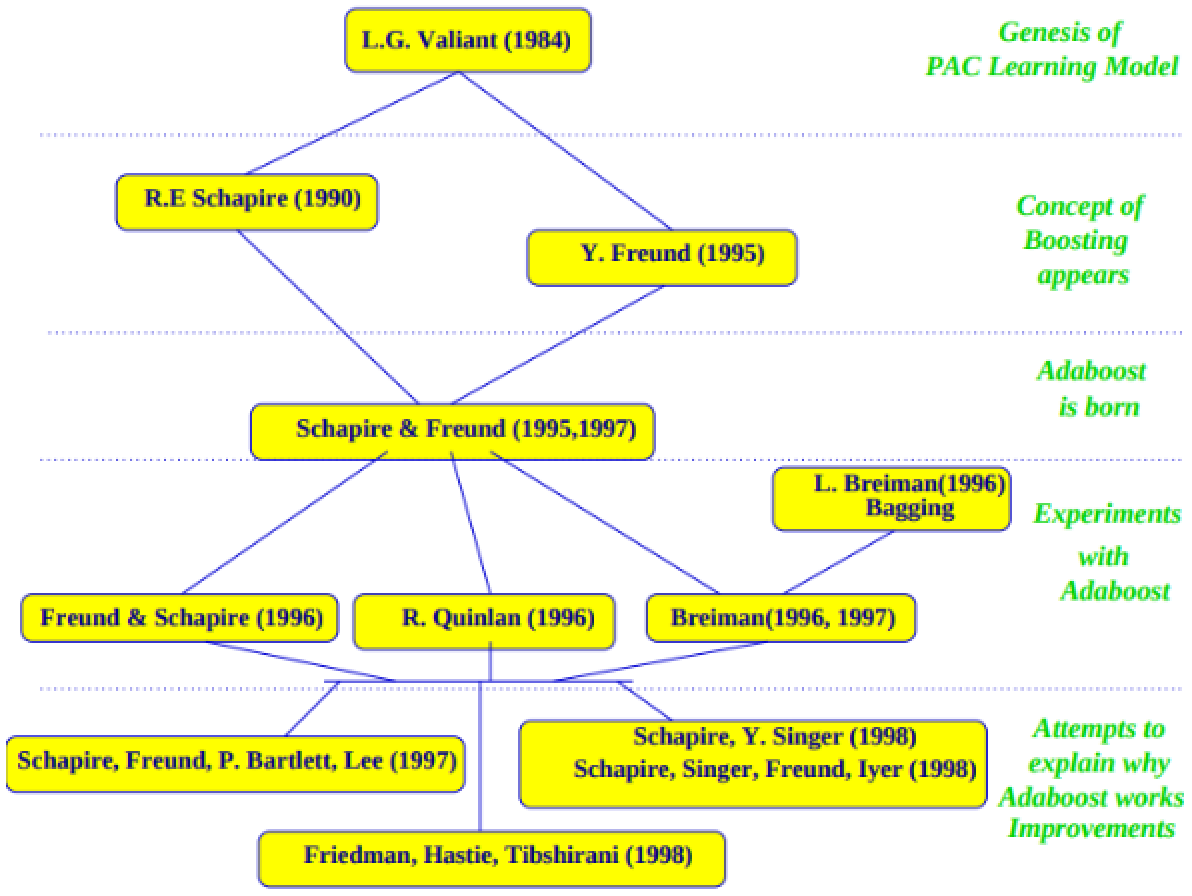

L.G. Valiant (1984)
Genesis of PAC Learning Model
R.E Schapire (1990)
Y. Freund (1995)
Concept of Boosting appears
Adaboost is born
Schapire & Freund (1995,1997)
L. Breiman(1996) Bagging
Experiments with Adaboost
Freund & Schapire (1996)
R. Quinlan (1996)
Breiman(1996, 1997)
Schapire, Freund, P. Bartlett, Lee (1997)
Schapire, Y. Singer (1998)
Schapire, Singer, Freund, Iyer (1998)
Attempts to explain why Adaboost works
Improvements
Friedman, Hastie, Tibshirani (1998)

Figure. History of Gradient Boosting

Image Source: https://web.stanford.edu/~hastie/TALKS/boost.pdf

Training the weak learners on the same sample set does prove to be futile. Hence, instead of manipulating the model, the underlying dataset is changed every iteration based on the feedback obtained at each iteration. This re-weighting strategy is a major difference that separates the AdaBoost algorithm from Gradient Boosting.

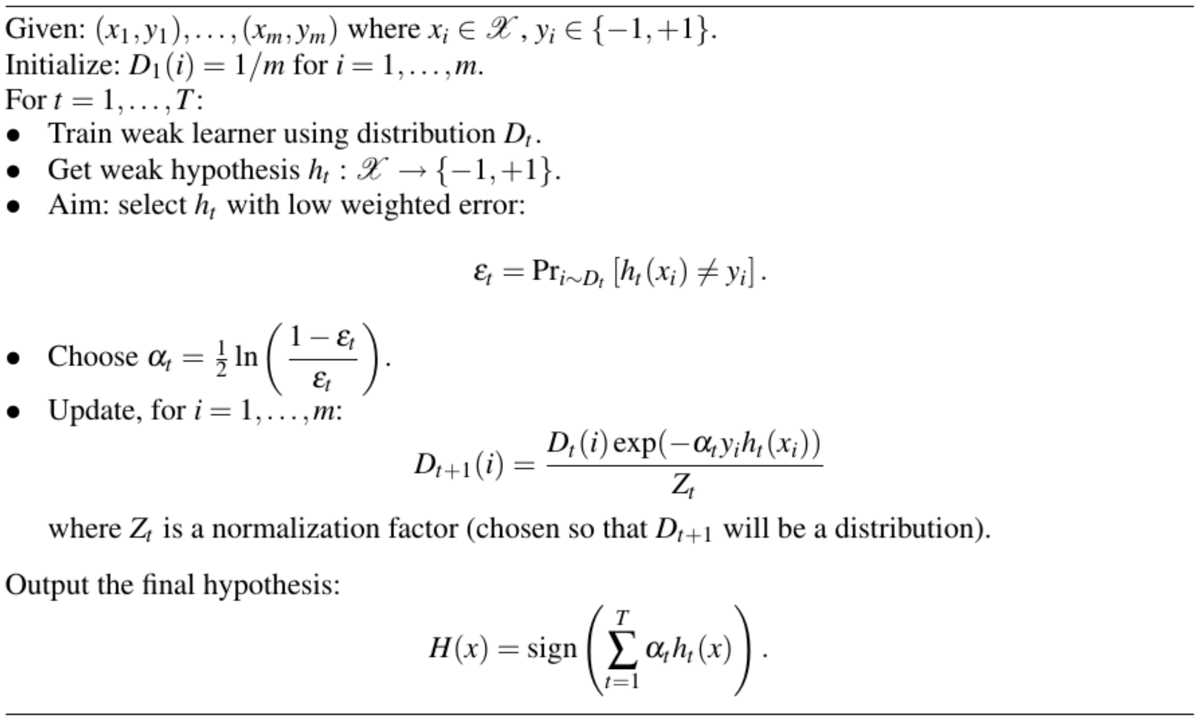
3. AdaBoost

AdaBoost is one of the earliest adaptive boosting algorithms which tunes its network parameters based on current iteration performance. The AdaBoost algorithm uses stumps (decision trees with a depth of 1) as its base-learners and creates *d* stumps with *d* being the number of features for each data point. The stump with the least Gini index is selected as the base learner and the same procedure is carried out iteratively. The new weights of each sample are calculated according to the following rule

$$new\ sample\ weight\ =\ sample\ weight\ \times e^{performance}$$

Each mis-classified sample is now being given more weight than the ones which have been correctly classified. This will be used to train the child-learners.

Formally, the algorithm has been defined as follows



```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathscr{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:
•  Train weak learner using distribution $D_t$.
•  Get weak hypothesis $h_t : \mathscr{X} \to \{-1, +1\}$.
•  Aim: select $h_t$ with low weighted error:
       $\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
•  Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$.
•  Update, for $i = 1, \ldots, m$:
       $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
   where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).
Output the final hypothesis:
       $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
```

Fig. 1 The boosting algorithm AdaBoost.

4. XGBoost

Also known as eXtreme Gradient Boosting, XGBoost is an implementation of gradient boosted decision trees specially optimized for performance. Gradient boosting machines are generally very slow due to the algorithm's sequential nature. For the sake of scalability, XGBoost focuses computational speed and model performance by providing -

- **Parallelization of tree construction** using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don't fit into memory.
- **Cache Optimization** of data structures and algorithms to make the best use of hardware.

The gradient boosting machines referred to here are covered in the subsequent section.

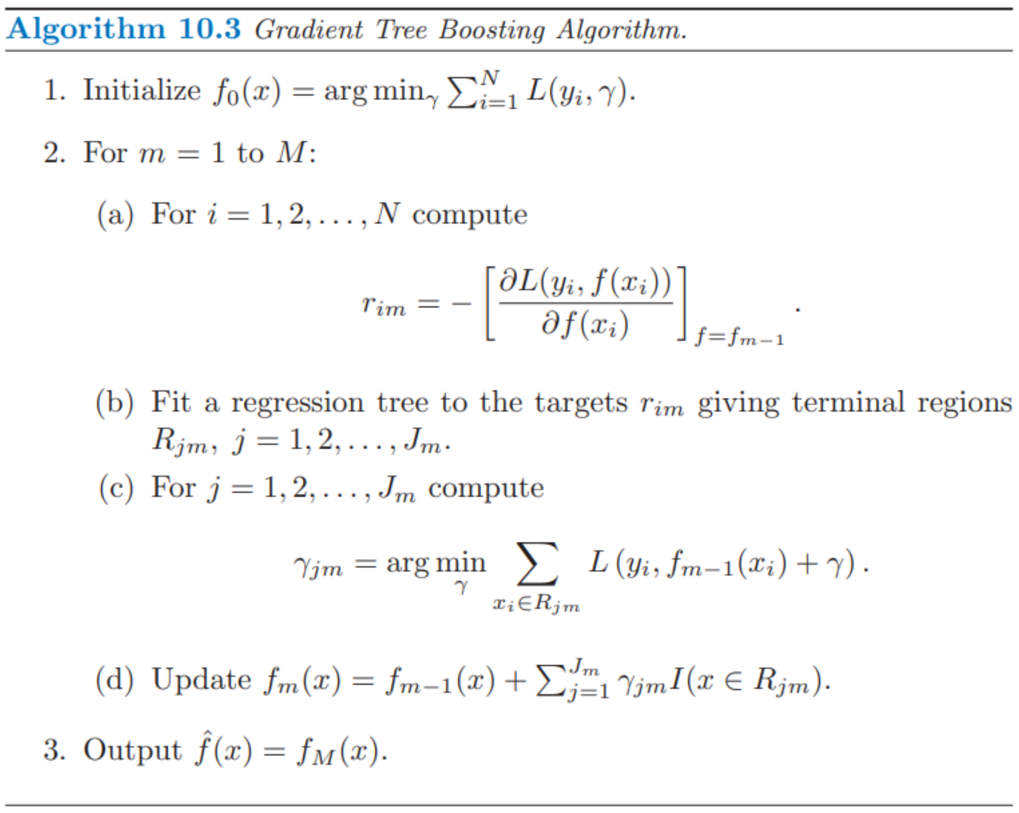
5. Gradient Boosting

5.1. Overview

In regression, gradient boosting constructs regression models by iteratively fitting weak learners to the pseudo-residuals of least squares at each iteration. The pseudo-residual is defined as the gradient of the loss function that is being minimized, w.r.t. values predicted by model for each training point evaluated at the current step. Similarly, in classification, a very similar process is repeated, except here we use a different loss function. The algorithms are more clearly described in section 3.2

5.2. Algorithm

5.2.1. Gradient Boosting



Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.
    (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions
        $R_{jm},\ j = 1, 2, \ldots, J_m$.
    (c) For $j = 1, 2, \ldots, J_m$ compute
            $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$.
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

5.2.2. Loss Function

5.2.2.1. Gradient Boosting Regression

We used squared loss function to train our model for the dataset

$$L(y_i, f(x_i)) = 1/2 * (y_i - f(x_i))^2$$

And calculated the residuals using

$$-\left[\frac{\delta L(y_i, f(x_i))}{\delta f(x_i)}\right]_{f(x) = f_{m-1}(x)} = y_i - f_{m-1}(x_i)$$

5.2.2.2. Gradient Boosting Classification

We used Logarithmic Loss function to train our model for the dataset

$$L(y_i, f(x_i)) =- (y_i * f(x_i) + log(1 + e^{f(x_i)})$$

Where f(x$_i$) is log of the odds

The corresponding residuals, in terms of log of odds

$$-[\frac{\delta L(y_i, f(x_i)}{\delta f(x_i)}]_{f(x) = f_{m-1}(x)} = y_i - (e^{log(odds)}/(1 + e^{log(odds)}))$$

Or in terms of predicted probability

$$-[\frac{\delta L(y_i, f(x_i)}{\delta f(x_i)}]_{f(x) = f_{m-1}(x)} = y_i - p_{predicted}$$

Steps Detailed

Step 1: Minimizing the loss function with constant predictor will give us

$$f_0(x) = log(odds_0)$$

Step 2: For m=1 to M

2.1 Calculate $-[\frac{\delta L(y_i, f(x_i)}{\delta f(x_i)}]_{f(x) = f_{m-1}(x)}$

which is just difference of observations - predicted_prob i.e

$$r_{im} = y_i - predicted_{i, m-1}$$

2.2 Fit a regression Tree to r_{im} with fixed depth to have similar weak learners

2.3 Now, we can have J terminal regions i.e., R_{jm} where j takes $1, 2,..., J$

We need to find γ_{jm} for each terminal node so that it can be added to come up with final model

$$\gamma_{jm} = \left(\sum_{x_i \in R_{jm}} r_{im}\right) / \left(\sum_{x_i \in R_{jm}} predicted_{i,m-1} * (1 - predicted_{i,m-1})\right)$$

2.4 Update Step

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} * I(x \in R_{jm})$$

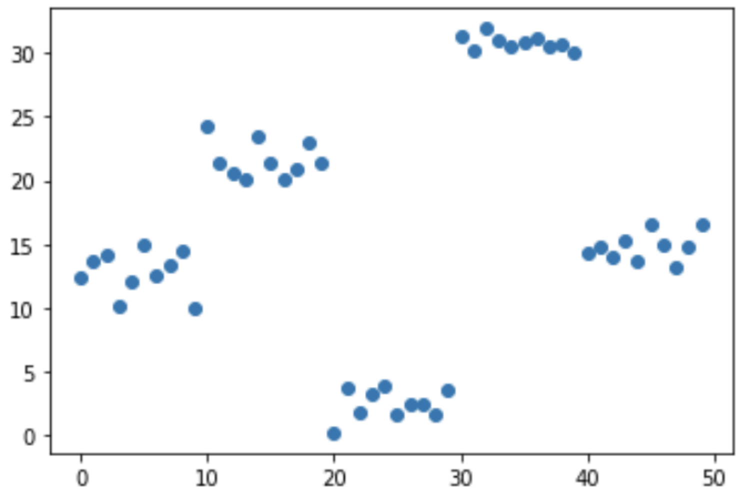
Step 3: Prediction Step

$$\hat{f}(x) = f_M(x)$$

5.3. Application on Simple Dataset

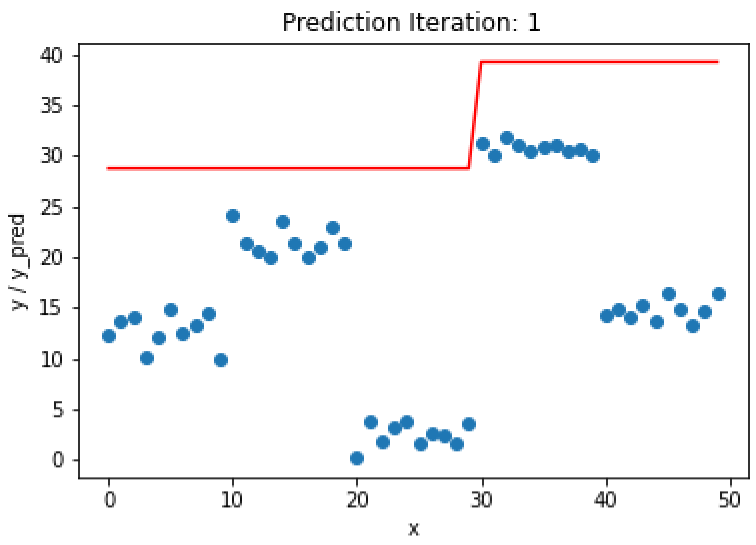
5.3.1. Gradient Boosting Regression

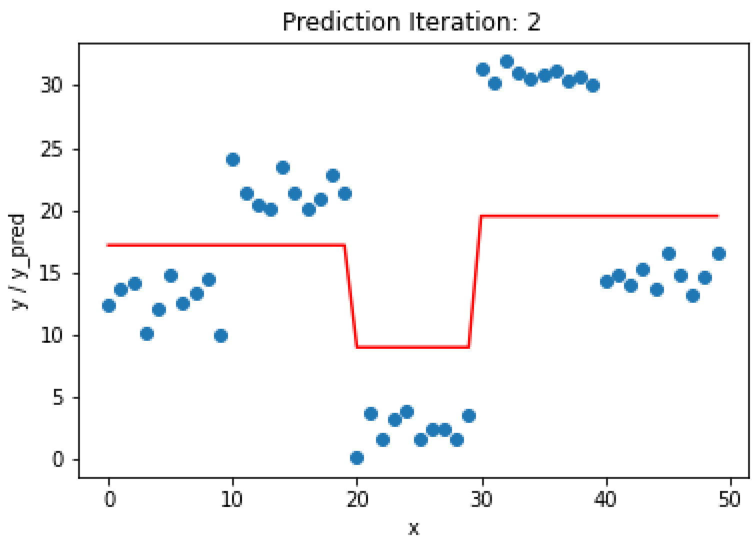
We apply the Gradient Boosting algorithm for a simple 2D regression dataset as shown below.



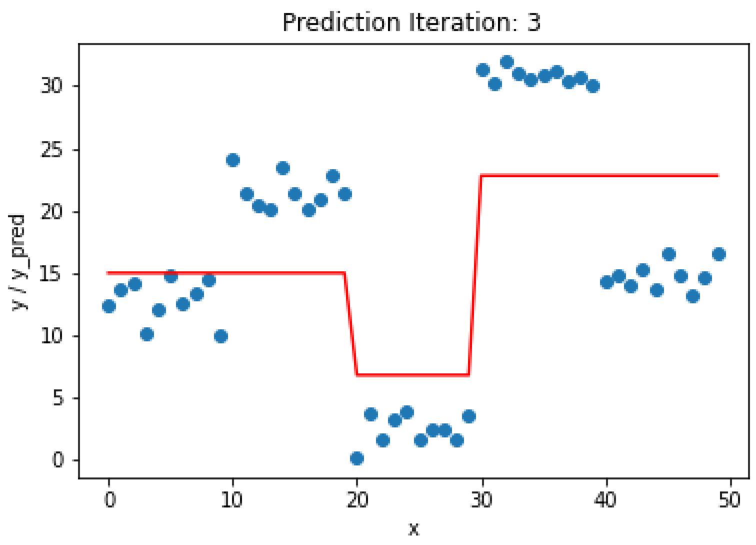
The gradient boosting algorithm, fits multiple decision trees iteratively. Decision tree, being a non-linear model, is able to fit non-linear decision boundaries to the data and hence is able to capture the underlying patterns. As observed, the model is able to represent the data points better in each iteration. Another observation is that, during the initial stages of the model, there is a definitive pattern in the residuals. Once the model learns the underlying representation, the residuals obtained are random and do not seem to possess any observable pattern.

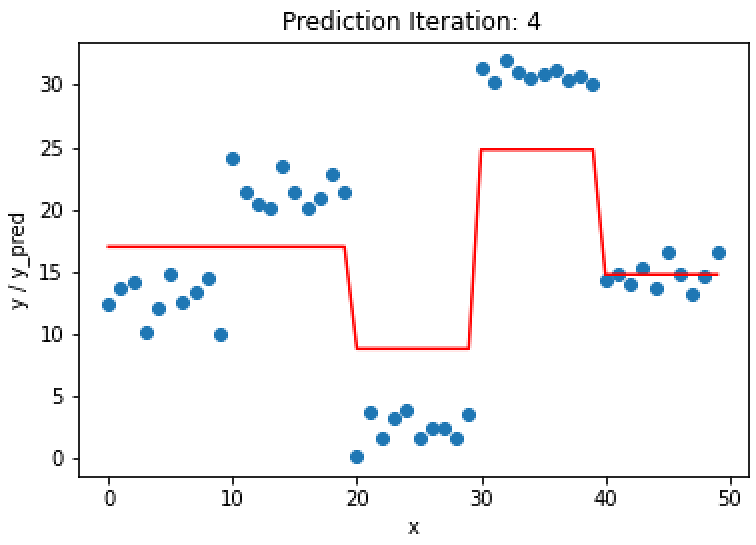
Iterative Visualization of Boosting Algorithm



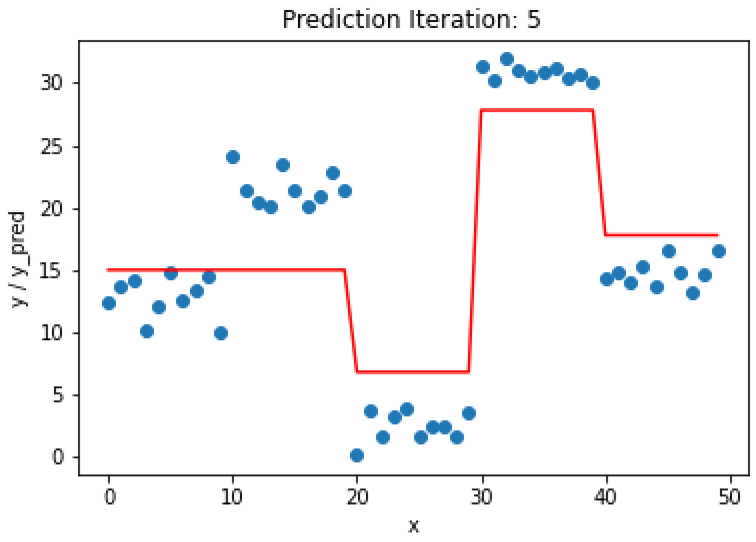


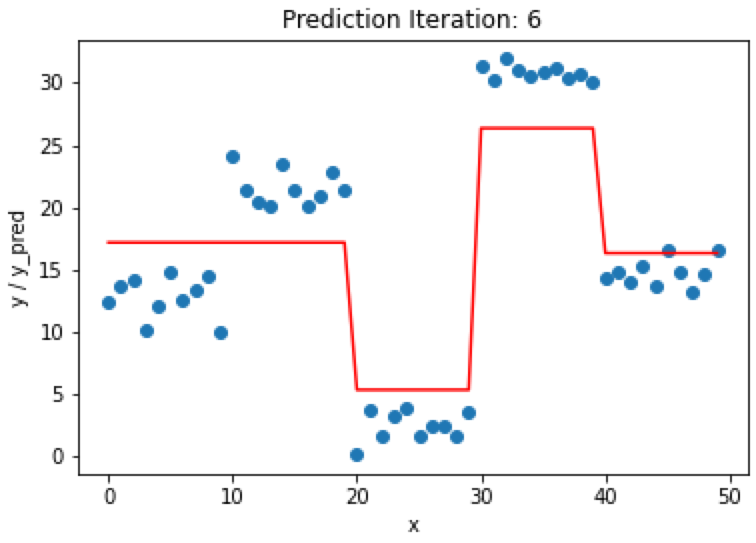
Code - https://github.com/svp19/gradient-boosting

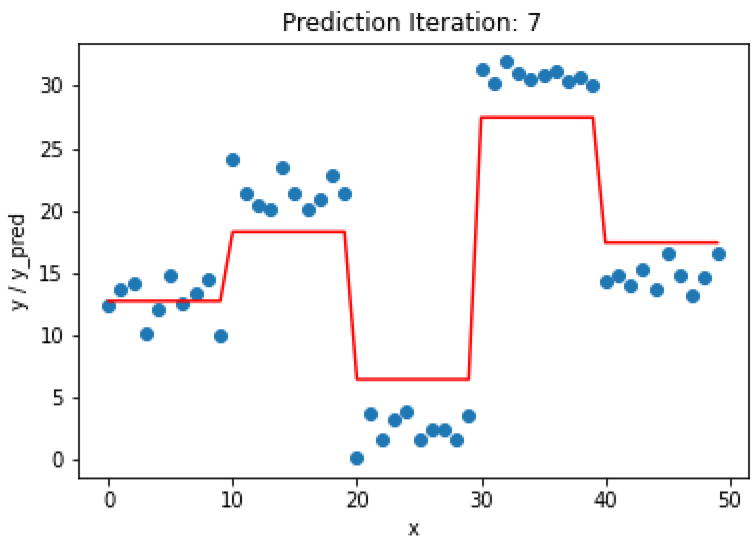


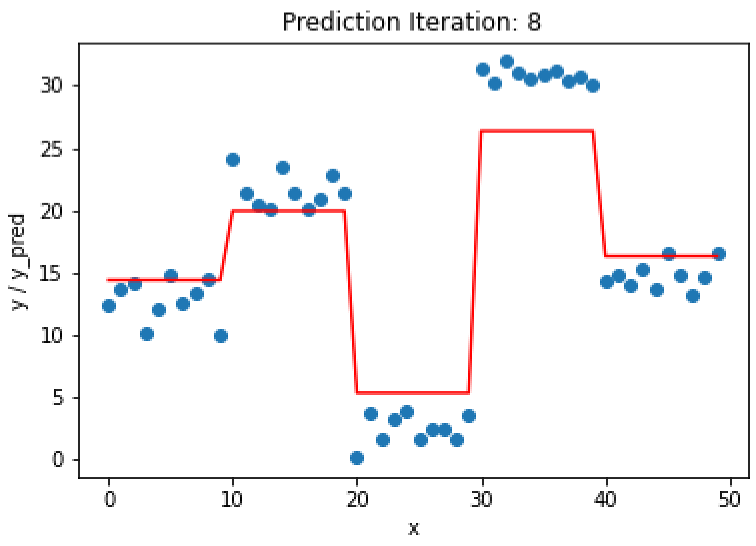


Code - https://github.com/svp19/gradient-boosting

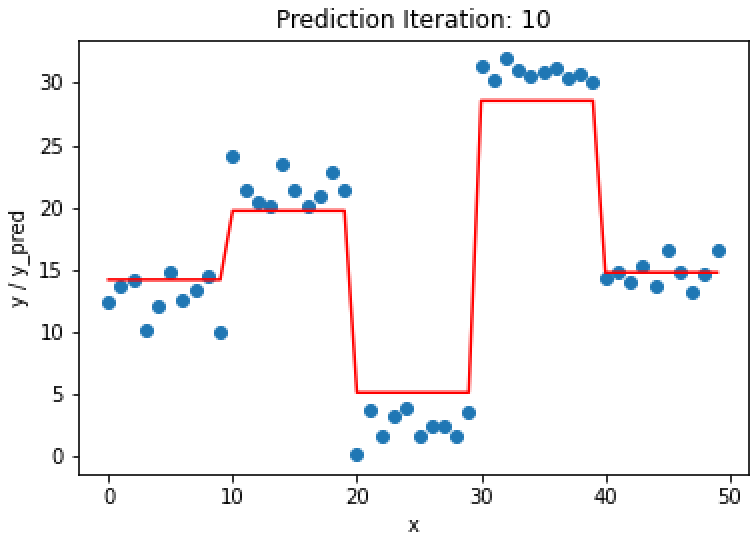

Prediction Iteration: 5
y / y_pred
x

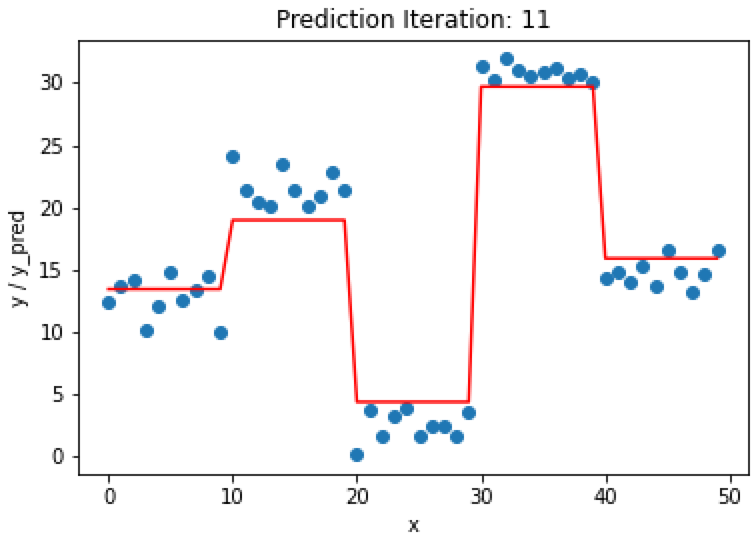

Prediction Iteration: 6
y / y_pred
x



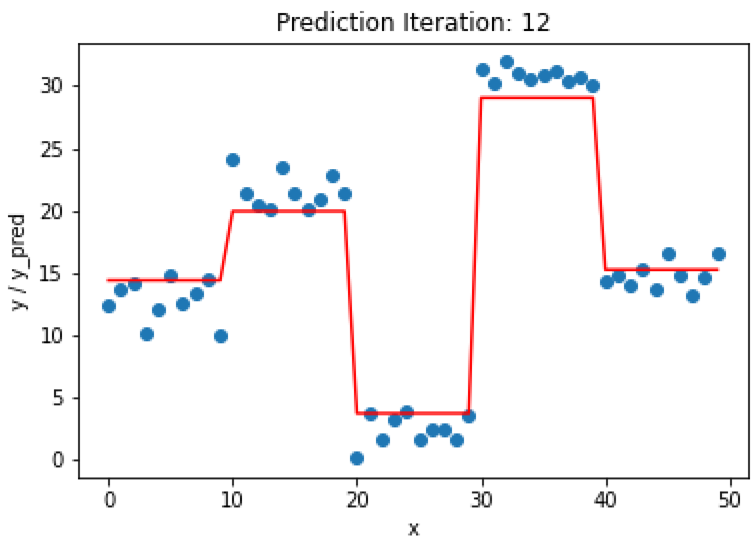


Code - https://github.com/svp19/gradient-boosting

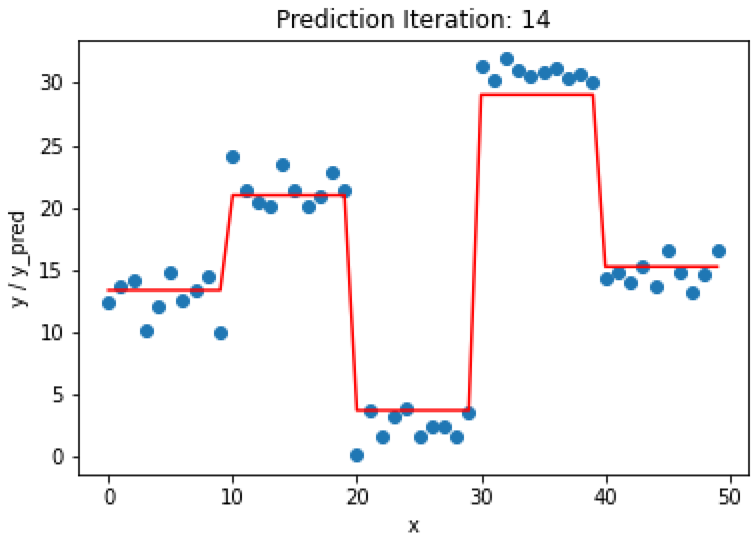


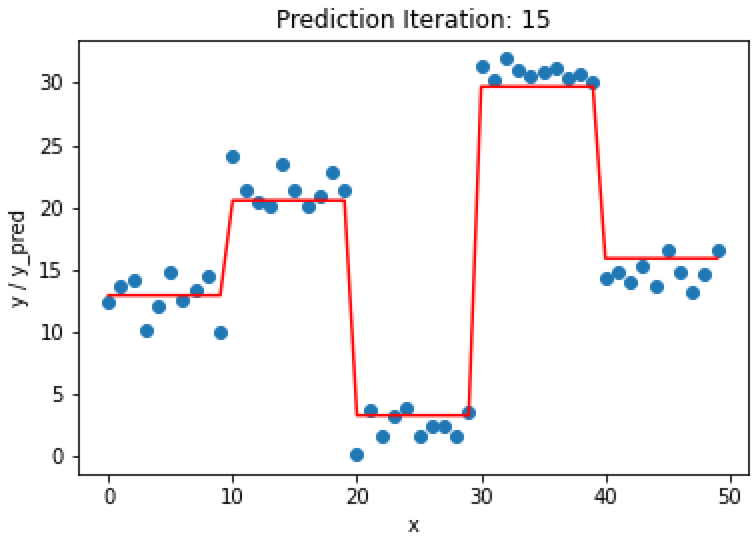


Code - https://github.com/svp19/gradient-boosting



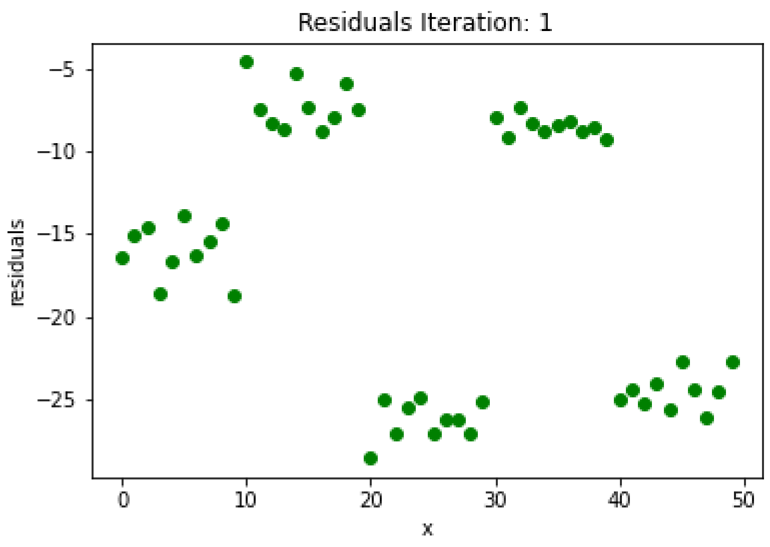
Code - https://github.com/svp19/gradient-boosting

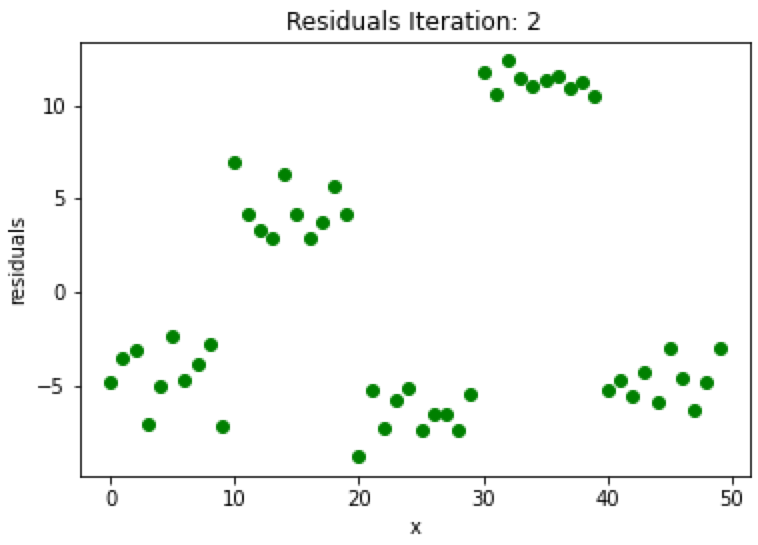




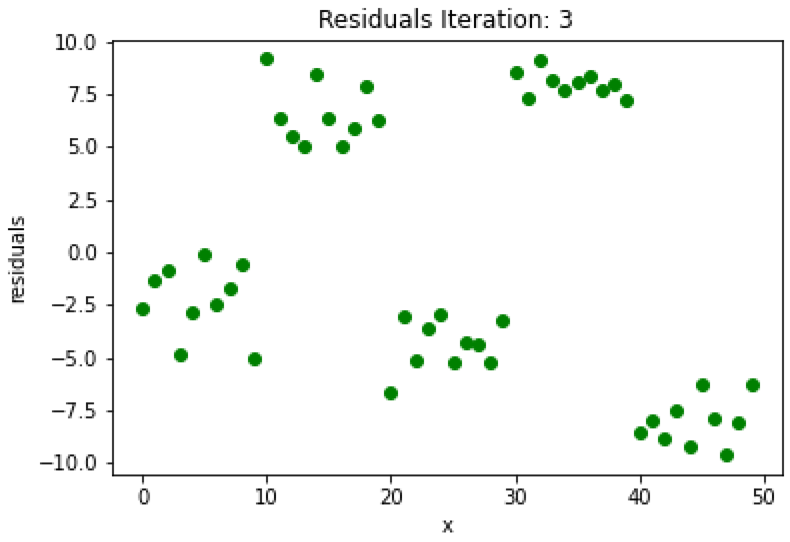
Code - https://github.com/svp19/gradient-boosting

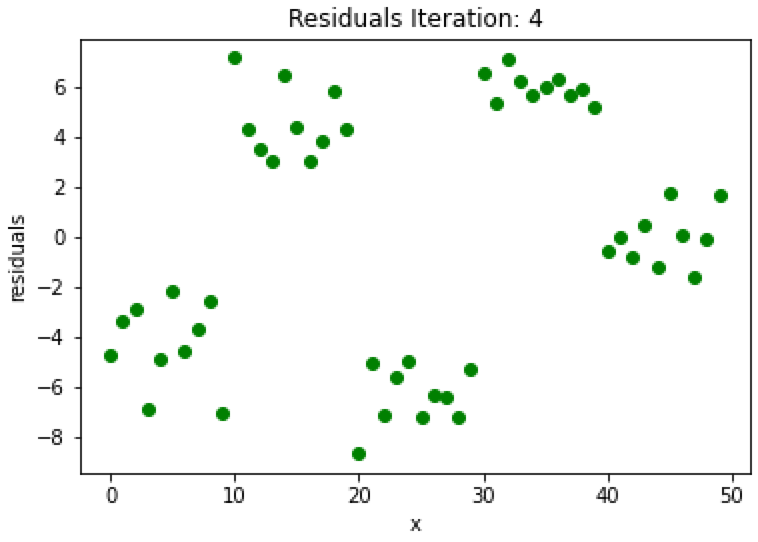
Iterative Visualization of Residuals while Boosting



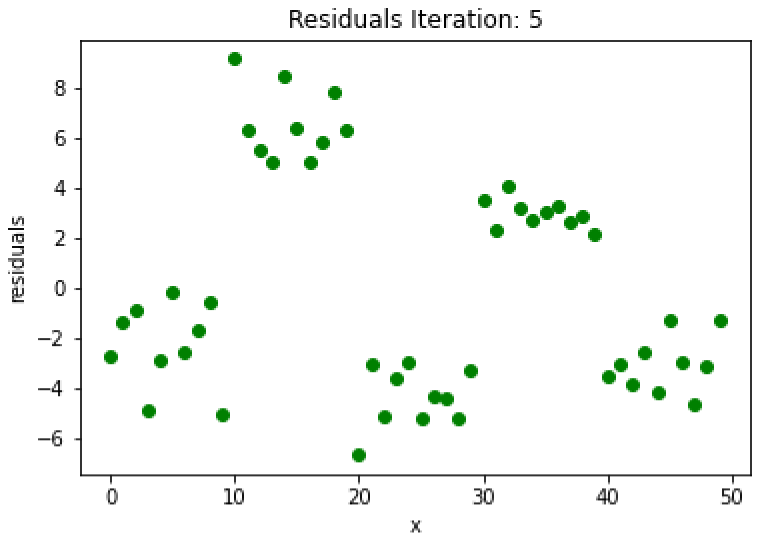


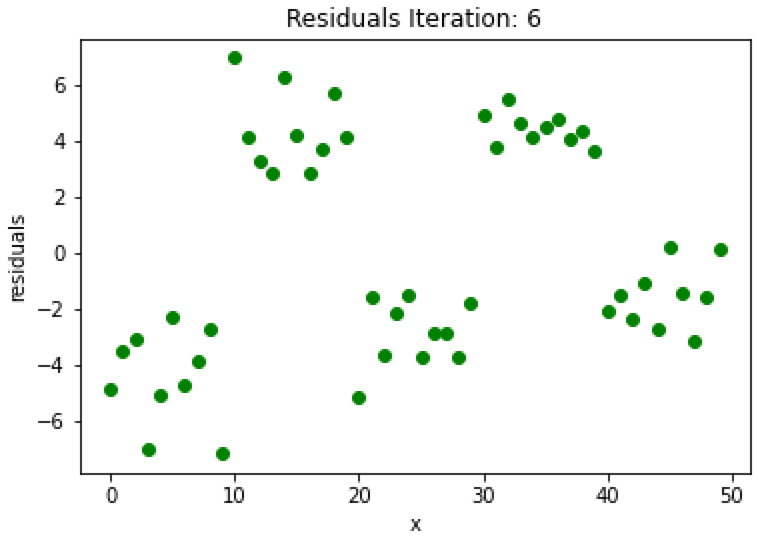
Code - https://github.com/svp19/gradient-boosting



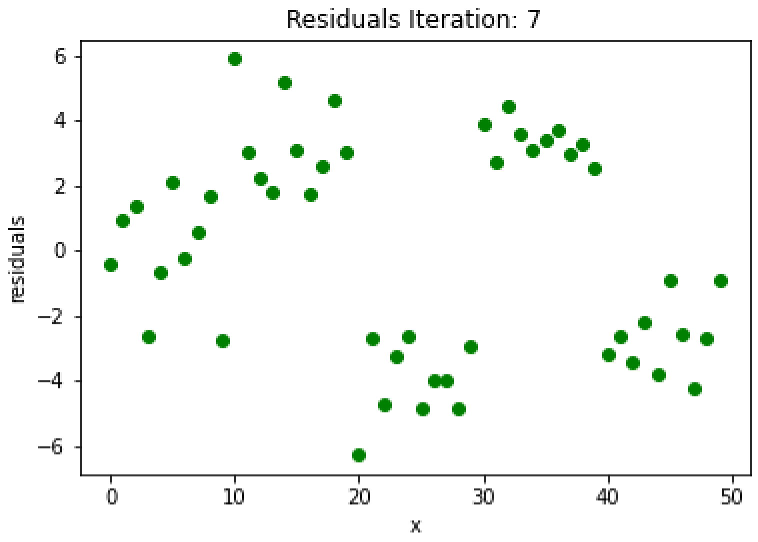


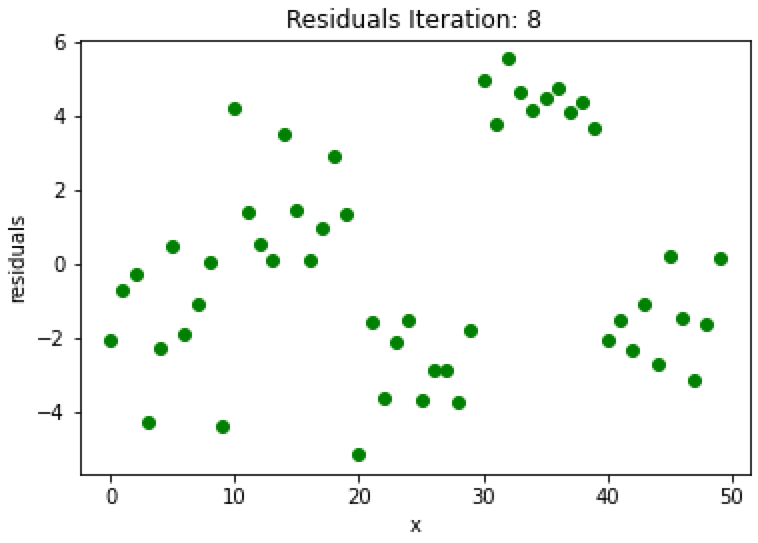
Code - https://github.com/svp19/gradient-boosting

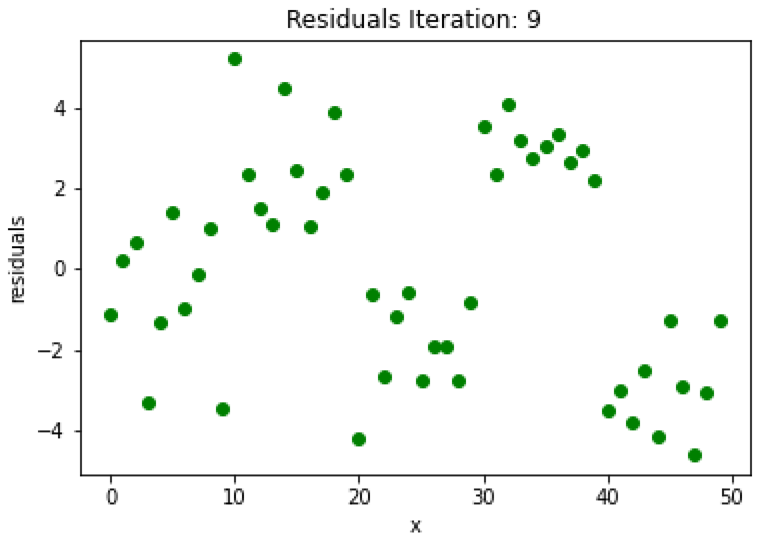


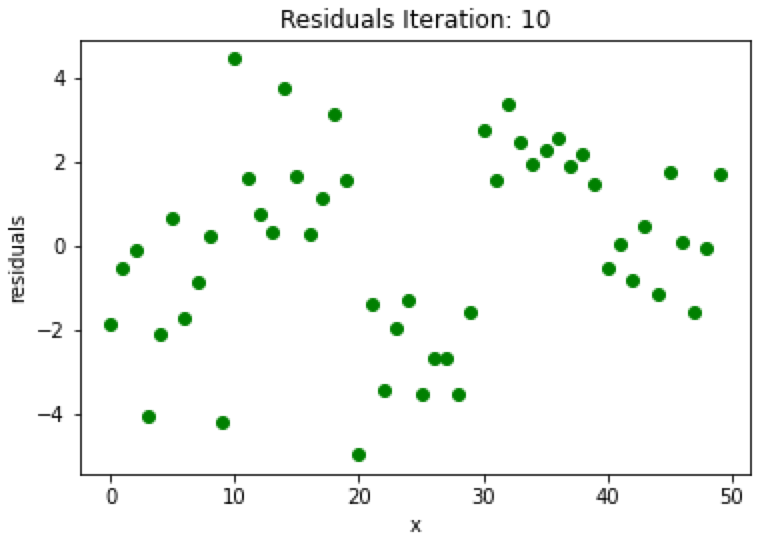


Code - https://github.com/svp19/gradient-boosting

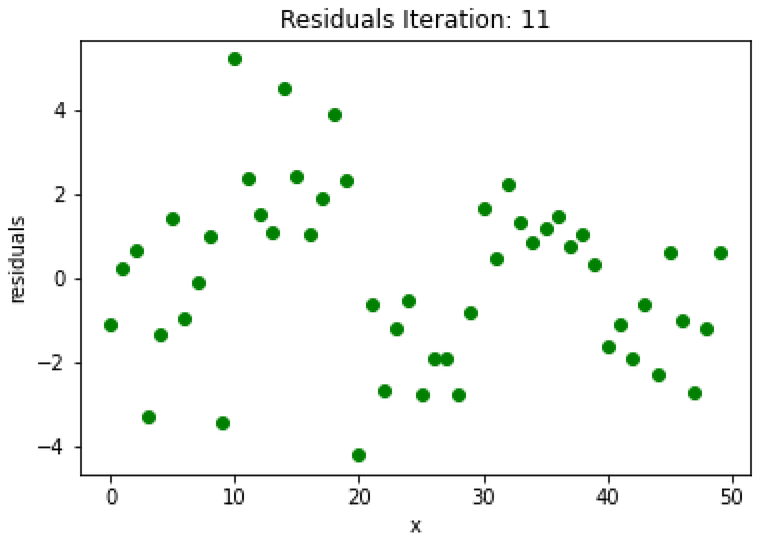

Residuals Iteration: 7
residuals
x

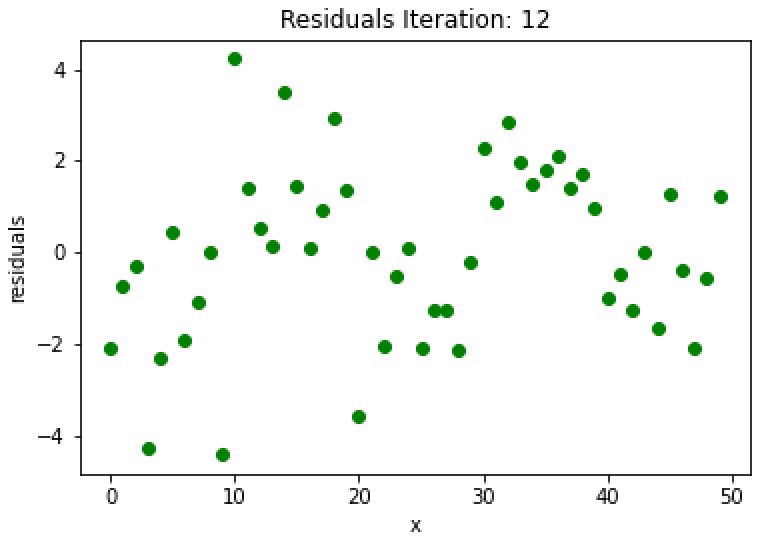

Residuals Iteration: 8
residuals
x



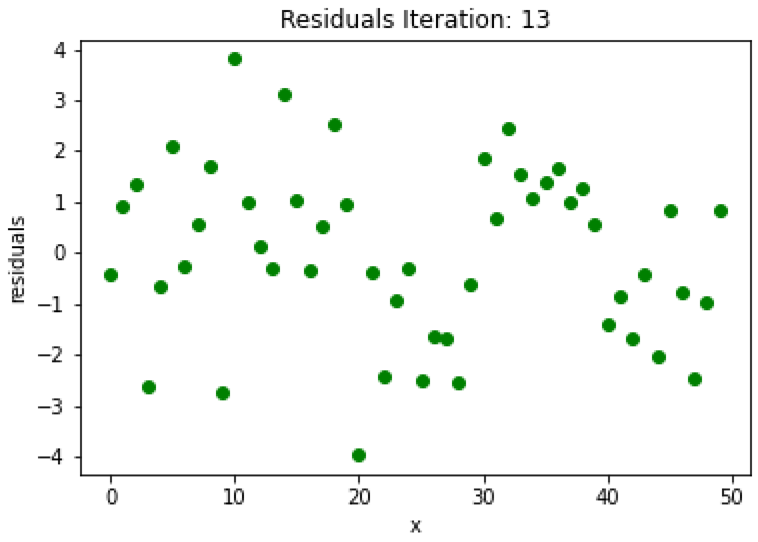


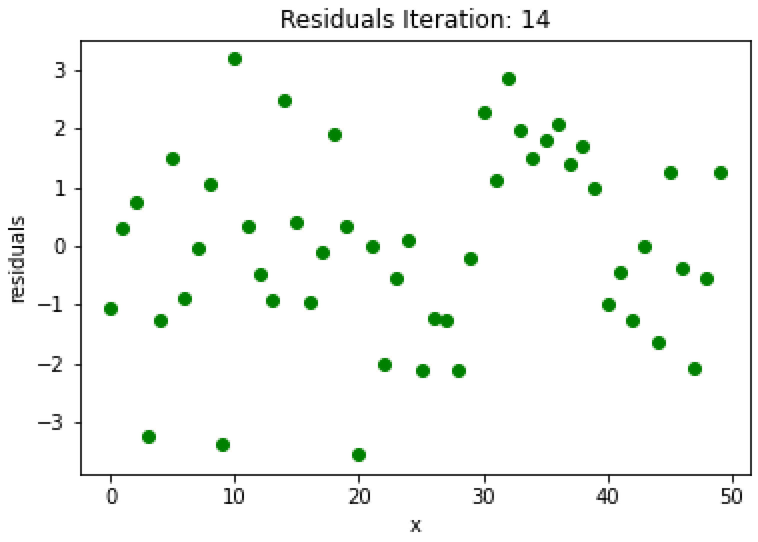
Code - https://github.com/svp19/gradient-boosting



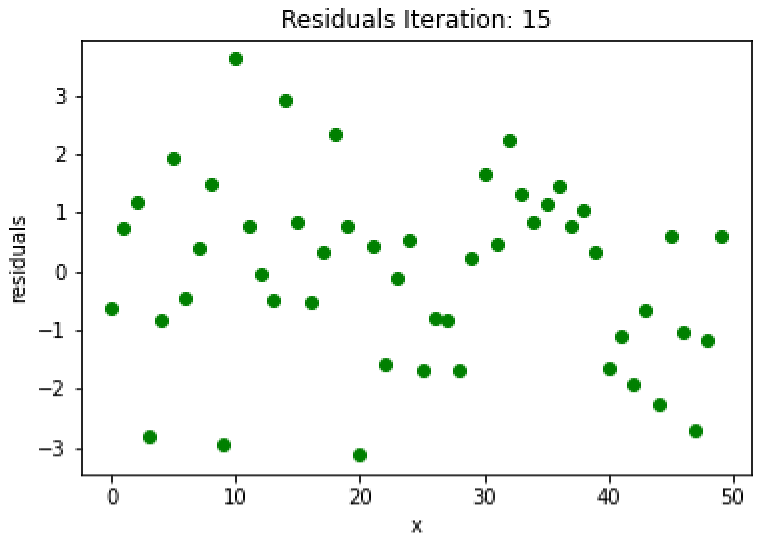


Code - https://github.com/svp19/gradient-boosting

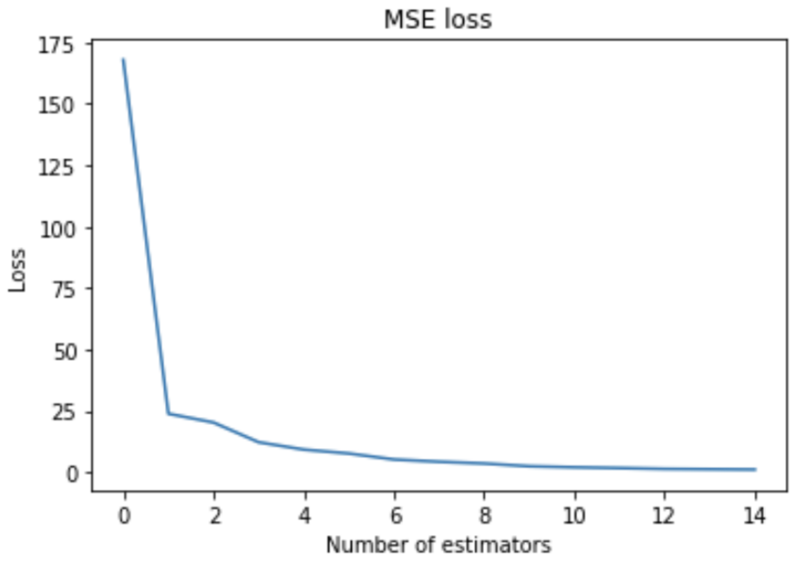




Code - https://github.com/svp19/gradient-boosting



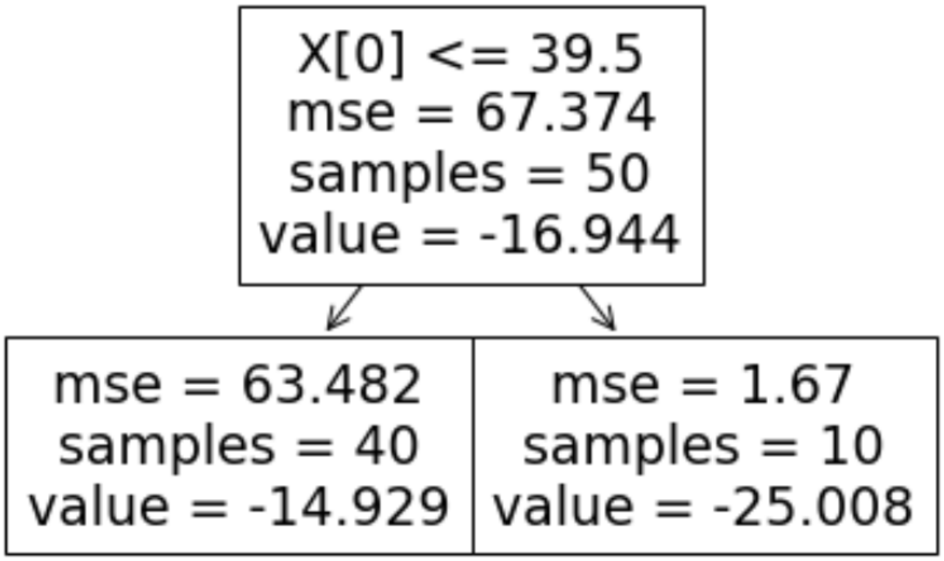
The train loss also reduces with increase in the number of iterations as observed



Code - https://github.com/svp19/gradient-boosting

5.3.1.1. Visualizing decision trees

Since the current dataset we are trying to fit the model on is fairly simple, to avoid overfitting we use a decision tree base-learner with a depth of 1. This decision tree tries to learn to classify samples into different leaves based on their feature representations. One such decision tree is shown below. Given a dataset of 50 points, the first stump segregates them into 40 and 10 based on a decision rule. Similarly the following weak-learners learn other decision boundaries based on feature values to obtain an optimal model.



5.3.1.2. Comparison of Loss Functions for Regression

- *Least squares loss function:*

$$L(y,\ h(x)) = \frac{1}{N} \times \sum_{i=1}^{N} (y_i - h(x_i))^2$$

- *Least absolute deviation loss function:*

$$L(y,\ h(x)) = \sum_{i=1}^{N} \left| y_i - h(x_i) \right|$$

- *Huber loss function:*

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Loss Function	**MAE**	**MSE**	**RMSE**
Least Squares	0.23	0.116	0.341
Least absolute deviation	0.237	0.172	0.415
Huber loss	0.235	0.133	0.365

Table: Comparison of losses and impact on performance for gradient boosting regression on simple dataset

5.3.2. Gradient Boosting Classifier

We applied Gradient Boosting Classification algorithm on the {0, 1} classification dataset.

5.3.2.1. Dataset

We created a custom small sized dataset to test our algorithm.

	Name	Attendance	Marks	Branch	loves_subject
0	Batman	1	10	ME	1
1	Superman	1	90	CSE	1
2	Flash	0	30	ME	0
3	Thanos	1	10	EE	0
4	Ironman	0	30	CSE	1
5	Hulk	0	50	ME	1

5.3.2.2. Results

Iteration 1:

```
Number of leaves  2
Indices for data [1 1 1 2 1 1]
for leaf 1, we have 5 related samples. and gamma is 0.6000000000000001
for leaf 2, we have 1 related samples. and gamma is -2.9999999999999996
```

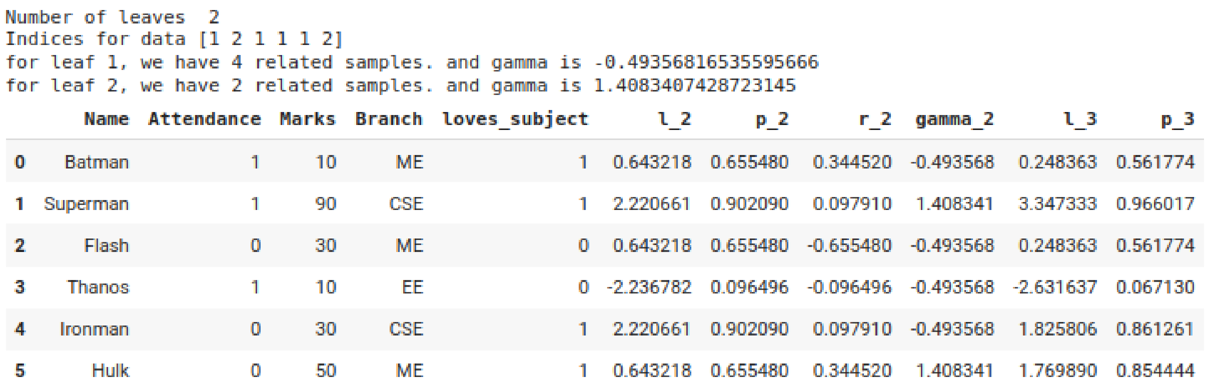
	Name	Attendance	Marks	Branch	loves_subject	l_0	p_0	r_0	gamma_0	l_1	p_1
0	Batman	1	10	ME	1	0.693147	0.666667	0.333333	0.6	1.173147	0.763713
1	Superman	1	90	CSE	1	0.693147	0.666667	0.333333	0.6	1.173147	0.763713
2	Flash	0	30	ME	0	0.693147	0.666667	-0.666667	0.6	1.173147	0.763713
3	Thanos	1	10	EE	0	0.693147	0.666667	-0.666667	-3.0	-1.706853	0.153572
4	Ironman	0	30	CSE	1	0.693147	0.666667	0.333333	0.6	1.173147	0.763713
5	Hulk	0	50	ME	1	0.693147	0.666667	0.333333	0.6	1.173147	0.763713

Iteration 2:

```
Number of leaves  2
Indices for data [1 2 1 1 2 1]
for leaf 1, we have 4 related samples. and gamma is -0.6624118798340858
for leaf 2, we have 2 related samples. and gamma is 1.3093916959030705
```

	Name	Attendance	Marks	Branch	loves_subject	l_1	p_1	r_1	gamma_1	l_2	p_2
0	Batman	1	10	ME	1	1.173147	0.763713	0.236287	-0.662412	0.643218	0.655480
1	Superman	1	90	CSE	1	1.173147	0.763713	0.236287	1.309392	2.220661	0.902090
2	Flash	0	30	ME	0	1.173147	0.763713	-0.763713	-0.662412	0.643218	0.655480
3	Thanos	1	10	EE	0	-1.706853	0.153572	-0.153572	-0.662412	-2.236782	0.096496
4	Ironman	0	30	CSE	1	1.173147	0.763713	0.236287	1.309392	2.220661	0.902090
5	Hulk	0	50	ME	1	1.173147	0.763713	0.236287	-0.662412	0.643218	0.655480

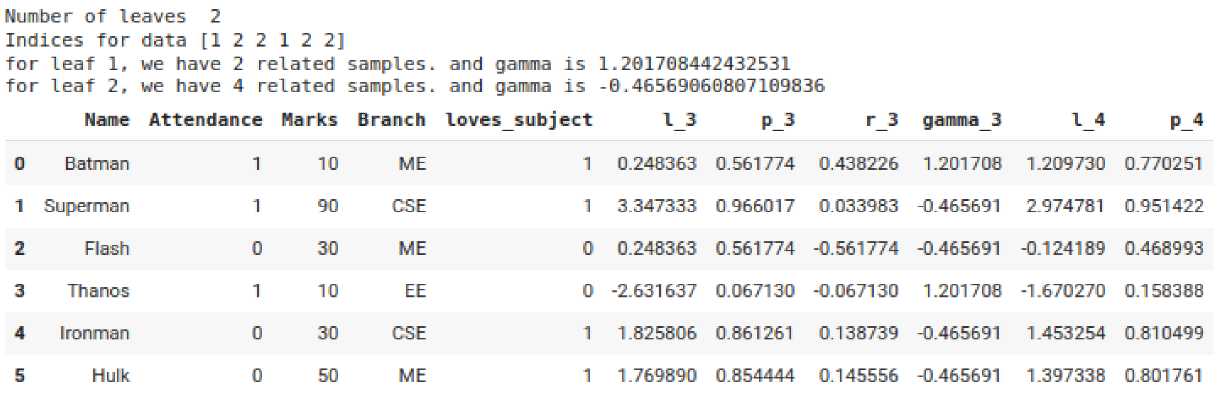
Iteration 3:



```
Number of leaves  2
Indices for data [1 2 1 1 1 2]
for leaf 1, we have 4 related samples. and gamma is -0.49356816535595666
for leaf 2, we have 2 related samples. and gamma is 1.4083407428723145
```

	Name	Attendance	Marks	Branch	loves_subject	l_2	p_2	r_2	gamma_2	l_3	p_3
0	Batman	1	10	ME	1	0.643218	0.655480	0.344520	-0.493568	0.248363	0.561774
1	Superman	1	90	CSE	1	2.220661	0.902090	0.097910	1.408341	3.347333	0.966017
2	Flash	0	30	ME	0	0.643218	0.655480	-0.655480	-0.493568	0.248363	0.561774
3	Thanos	1	10	EE	0	-2.236782	0.096496	-0.096496	-0.493568	-2.631637	0.067130
4	Ironman	0	30	CSE	1	2.220661	0.902090	0.097910	-0.493568	1.825806	0.861261
5	Hulk	0	50	ME	1	0.643218	0.655480	0.344520	1.408341	1.769890	0.854444

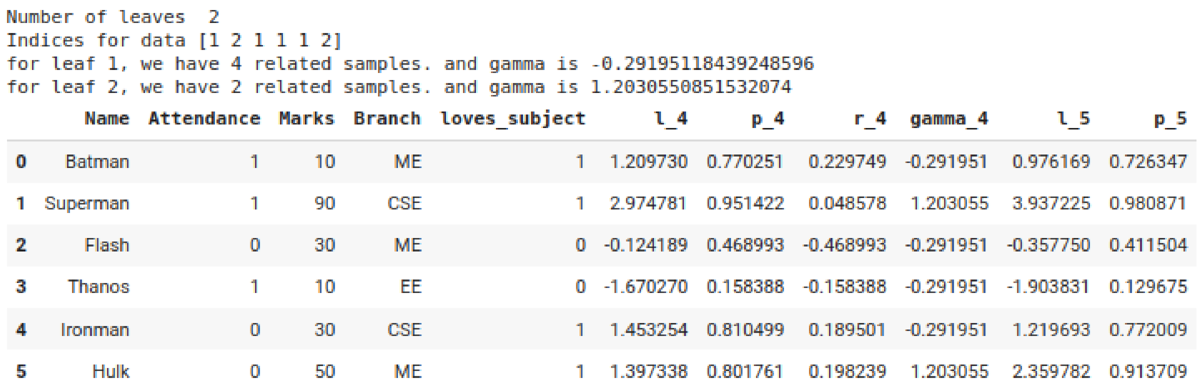
Iteration 4:



```
Number of leaves  2
Indices for data [1 2 2 1 2 2]
for leaf 1, we have 2 related samples. and gamma is 1.201708442432531
for leaf 2, we have 4 related samples. and gamma is -0.46569060807109836
```

	Name	Attendance	Marks	Branch	loves_subject	l_3	p_3	r_3	gamma_3	l_4	p_4
0	Batman	1	10	ME	1	0.248363	0.561774	0.438226	1.201708	1.209730	0.770251
1	Superman	1	90	CSE	1	3.347333	0.966017	0.033983	-0.465691	2.974781	0.951422
2	Flash	0	30	ME	0	0.248363	0.561774	-0.561774	-0.465691	-0.124189	0.468993
3	Thanos	1	10	EE	0	-2.631637	0.067130	-0.067130	1.201708	-1.670270	0.158388
4	Ironman	0	30	CSE	1	1.825806	0.861261	0.138739	-0.465691	1.453254	0.810499
5	Hulk	0	50	ME	1	1.769890	0.854444	0.145556	-0.465691	1.397338	0.801761

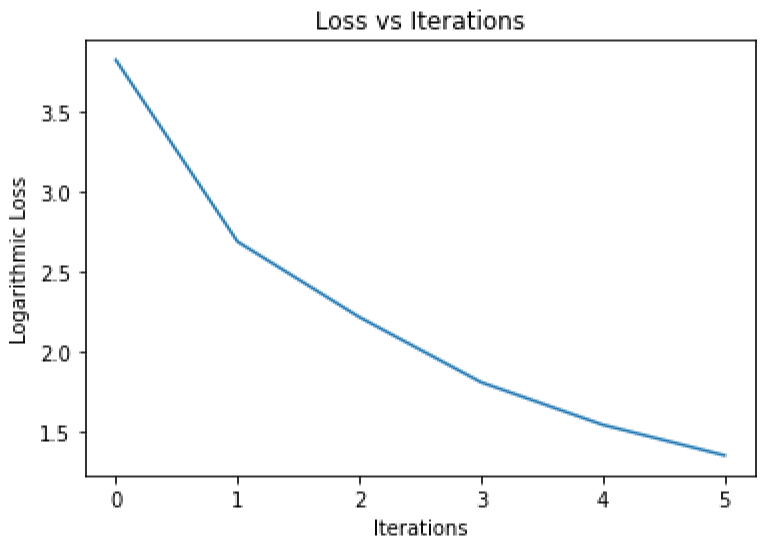
Iteration 5:



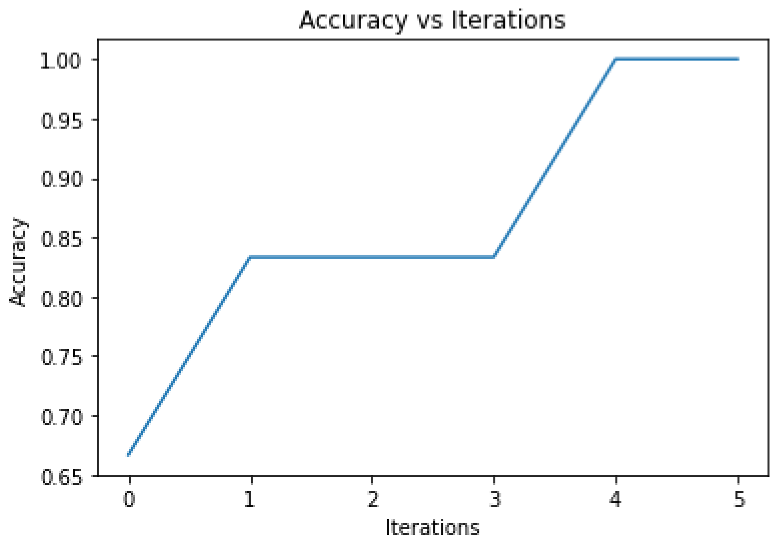
```
Number of leaves  2
Indices for data [1 2 1 1 1 2]
for leaf 1, we have 4 related samples. and gamma is -0.29195118439248596
for leaf 2, we have 2 related samples. and gamma is 1.2030550851532074
```

	Name	Attendance	Marks	Branch	loves_subject	l_4	p_4	r_4	gamma_4	l_5	p_5
0	Batman	1	10	ME	1	1.209730	0.770251	0.229749	-0.291951	0.976169	0.726347
1	Superman	1	90	CSE	1	2.974781	0.951422	0.048578	1.203055	3.937225	0.980871
2	Flash	0	30	ME	0	-0.124189	0.468993	-0.468993	-0.291951	-0.357750	0.411504
3	Thanos	1	10	EE	0	-1.670270	0.158388	-0.158388	-0.291951	-1.903831	0.129675
4	Ironman	0	30	CSE	1	1.453254	0.810499	0.189501	-0.291951	1.219693	0.772009
5	Hulk	0	50	ME	1	1.397338	0.801761	0.198239	1.203055	2.359782	0.913709

Logarithmic Loss vs Iterations



Accuracy vs Iterations

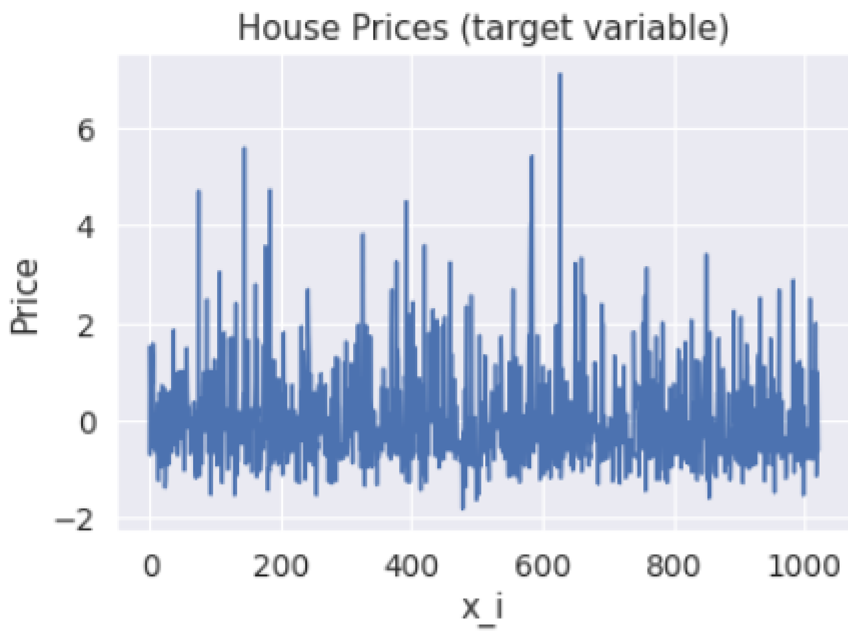


5.4. Application on House Price Regression

5.4.1. Dataset

The Ames Housing dataset was compiled by Dean De Cock for use in data science education. It is a modernized and expanded version of the often cited Boston Housing dataset. With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, the challenge is to predict the final price of each home.

There is no clear pattern in the target prices as seen in the below graph.



Link to dataset - https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

5.4.2. Preprocessing

There are a lot of missing values in the dataset. We drop columns with too many missing values. For the other columns with lesser missing values we impute that column with the mean of the values in it. We next pick the top 15 features based on their correlation with the target variable. The final dataset is of size (1460, 14) and this dataset is then divided into 70:30 train-test splits.

5.4.3. Results

We compare our implementation of the gradient boosting algorithm against sklearn's implementation. We also demonstrate its performance compared to other famous regression techniques.

Sl No.	Model	MAE	MSE	RMSE	Average Bias	Average Variance
1.	Linear Regression	0.2910	0.2999	0.5477	0.299	0.006
2.	DecisionTreeRegressor	0.3327	0.2297	0.4793	0.161	0.162
3.	SupportVector Regression	0.2340	0.1899	0.4358	0.196	0.010
4.	RandomForest Regressor	0.2359	0.1518	0.3896	0.167	0.024
5.	LightGBM	0.2455	0.1586	0.3982	0.151	0.023
6.	AdaBoostRegressor	0.3134	0.1978	0.4447	0.187	0.037
7.	XGBoostRegressor	0.2398	0.1309	0.3618	0.143	0.051
8.	GradientBoostingRegressor *(*ours)*	0.2609	0.1444	0.3800	-	-
9.	GradientBoostingRegressor	0.2299	0.1156	0.3400	0.143	0.026

*MAE - mean absolute error

*MSE - mean squared error

*RMSE - root mean squared error

Average Bias and Variance were calculated with the help of the mlxtend library.

5.5. Application on Titanic Dataset

5.5.1. Dataset

The sinking of the Titanic is one of the most infamous shipwrecks in history. Unfortunately, there weren't enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. In this dataset, the task is to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc).

Distribution of Target Variable - Survived

0 :	549
1 :	342

5.5.2. Preprocessing

The dataset has 10 features. Of these, 'Cabin' and 'Ticket' are dropped due to too many missing values in them. We create a new feature called 'Title' (example - Major, Capt.) that is extracted from the name. All categorical features are label encoded. The other features with missing values are either imputed with the mean or median of the column. The final dataset is of size (891, 8) and this dataset is then divided into 80:20 train-test splits.

5.5.3. Results

Sl No.	Model	Train Accuracy	Test Accuracy	Average Bias	Average Variance
1.	Logistic Regression	0.7963	0.7988	0.196	0.040
2.	DecisionTreeClassifier	0.8722	0.8268	0.173	0.088
3.	SupportVectorClassifier	0.7823	0.7821	0.218	0.036
4.	RandomForestClassifier	0.8441	0.8156	0.168	0.054
5.	KNeighboursClassifier	0.8539	0.7988	0.190	0.101
6.	Gaussian Naive Bayes	0.7275	0.7150	0.291	0.131
7.	Bernoulli Naive Bayes	0.7851	0.7821	0.218	0.018
8.	Multinomial Naive Bayes	0.7359	0.7486	0.251	0.026
9.	AdaBoostClassifier	0.8174	0.7877	0.207	0.060
10.	XGBoostClassifier	0.8693	0.8324	0.173	0.082
11.	GradientBoostingClassifier*(*ours)*	0.81	0.8089	-	-
12.	GradientBoostingClassifier	0.8721	0.8268	0.179	0.086

5.6. Improving the Boosting Models

The Gradient Boosting algorithm is greedy in nature and prone to overfit eventually. To counter this there are some techniques that maybe used -

5.6.1. Constraining the Weak Learner

Strong learners are more prone to overfit faster. It is often observed that having more multiple weak learners deliver better results. By reducing the model complexity of the decision tree we can ensure it behaves as a weak learner. Here are few parameters of the decision tree that can be tweaked -

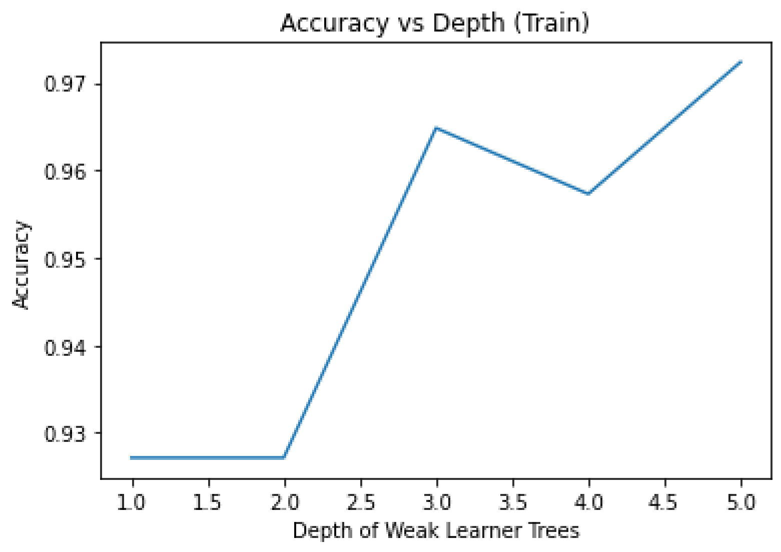
Number of trees - Higher the number of trees, more slowly, does the model overfit. Ideally, trees are added until no further improvement is observed.

Number of nodes or number of leaves - constrains the size of the tree.

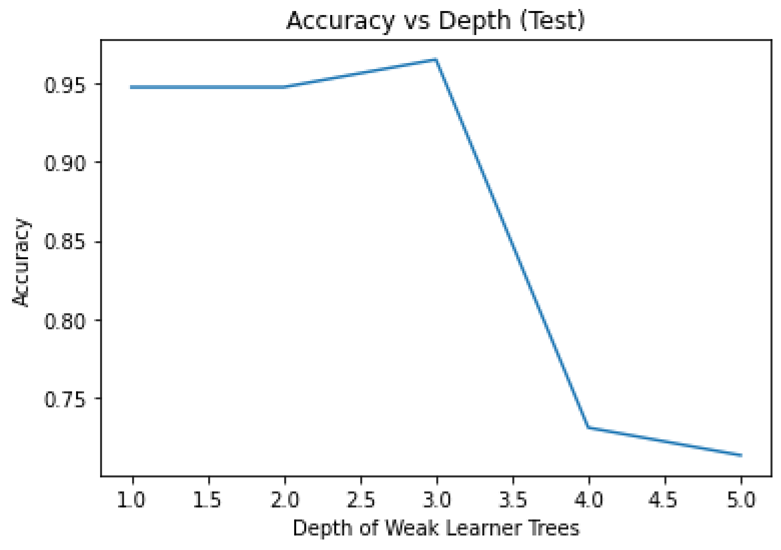
Tree depth - model complexity increases with depth and very deep trees may overfit much sooner.

Number of observations per split - is the minimum amount of training data at a training node before a split can be considered

Train Accuracy vs Depth



Test Accuracy vs Depth



5.6.2. Random Sampling

The idea of allowing trees to be created from subsamples of the data which was seen in random forests and other bagging techniques can also be extended here. This version, also known as ***stochastic gradient boosting,*** often leads to improved results depending on the data.

5.6.3. Shrinkage

Another important parameter to tweak is the learning rate or the shrinkage as it is often used to weight the contributions of each tree with a weight value less than 1.

Table. Results of gradient boosting on the House Prices Regression dataset by varying the learning rate.

Sl. No.	Learning Rate	Number of estimators	MAE	MSE	RMSE	Average Bias	Average Variance
1.	0.01	500	0.2276	0.1166	0.3414	0.143	0.026
2.	0.05	100	0.2299	0.1152	0.3394	0.143	0.026
3.	0.1	100	0.2259	0.1088	0.3299	0.137	0.030

5.6.4. Regularization

The leaves of the decision tree regressor can be thought of as the weights of the network. We can regularize using the l1-norm of l2-norm of these leaf values to improve model generalization.

5.7. Advantages of Gradient Boosting

Step by Step Approach

Boosting focuses step by step on difficult examples that give a nice strategy to deal with unbalanced datasets by strengthening the impact of the positive class.

Optimization in Function Space

It performs the optimization in function space (rather than in parameter space) which makes the use of custom loss functions much easier.

Not Much Data Preprocessing Required

It often works great with categorical and numerical values without preprocessing.

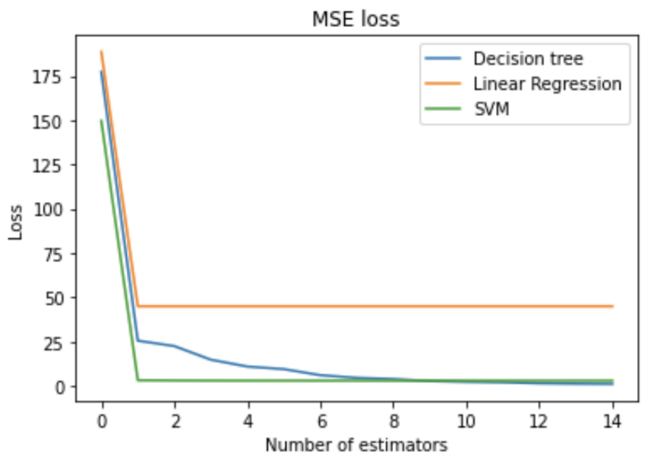
For better Generalization

Learning rate and depth of the tree can be made low to reduce overfitting.

5.8. Disadvantages of Gradient Boosting

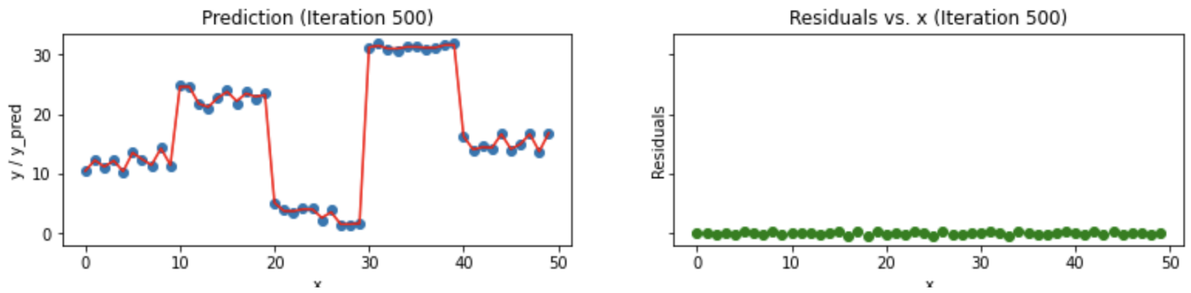
The gradient boosting algorithm primarily reduces the bias in model predictions and hence is susceptible to high variance or overfitting issues. Model overfitting is majorly caused due to two problems

- **Considerably strong learners are used instead of weak learners**
  In this case, each individual model fits the data considerably well leading to overfitting issues in very minimal iterations



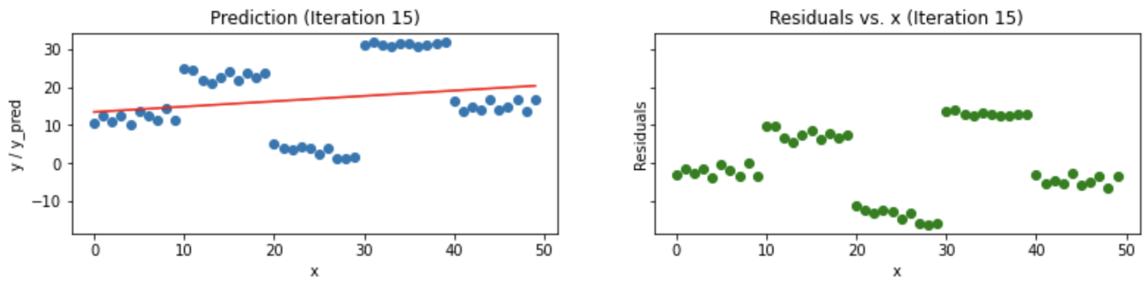
Since SVM is a stronger learner as compared to Decision trees or Linear Regression models. Hence, the train loss rapidly descends in case of the SVM classifier which leads to overfitting.

- **Using a very large number of weak learners**. In this case, the weak learners in later iterations fit the noise in the training data as well hence leading to overfitting. As observed in the below figure, using a very large number of estimators leads to overfitting since the model memorizes each point and will not be able to generalize well and all the residuals are at 0.



5.9. Linear models as weak learners for regression models

While considering a basic linear regression as a weak learning model for a non-linear dataset, we observe that the gradient boosting model predicts the best fit line itself onto the data. Since after certain iterations, the predicted linear decision boundary over the residuals does not change and hence, there is no improvement in the model over further iterations. Hence the MSE does not fall below a certain level.



5.10. Comparison between AdaBoost and Gradient Boosting

Sl No.	AdaBoost	Gradient Boosting
1.	Limitations of previous model are overcome by weighted data points.	Limitations of previous model are overcome by identifying the gradients.
2.	Both classifiers and observations are weighted thus capturing maximum variance.	All classifiers are equally weighted by the learning rate specified.

6. Conclusion

Gradient Boosting is an effective technique that can be used to iteratively convert weak learners to a strong one. We explore the basic intuition and formulation of the algorithm. We demonstrate the efficacy of the technique on a regression problem - House Prices, and we observe that it performs much better than most other popular machine learning algorithms, giving lower bias and nearly the same or higher variance. Similarly, the model performs well on the Titanic dataset with an accuracy of 82.68%. We also show how the model performance can be improved by constraining the weak learner, random sampling, regularization and choosing a good learning rate. The algorithm does not work well when linear models are used as weak learners but proves to be effective when models like decision trees are used. Finally, gradient boosting comes with its own pros and cons and while it does often give us a model that generalizes well; is also prone to overfit.

7. References

[1] The Evolution of Boosting Algorithms

[2] Greedy Function Approximation: A Gradient Boosting Machine Jerome H. Friedman* IMS 1999 Reitz Lecture February 24,1999 (modified

[3] A Gentle Introduction to Gradient Boosting

[4] Gradient Boosting

[5] The Elements of Statistical Learning

[6] Experiments with a New Boosting Algorithm

[7] A Short Introduction to Boosting

[8] L. G. Valiant. A theory of the learnable. Communications of the ACM, 27(11):1134–1142, November 1984.

[9] Michael J. Kearns and Umesh V. Vazirani. An Introduction to Computational Learning Theory. MIT Press, 1994.

[10] Michael Kearns and Leslie G. Valiant. Learning Boolean formulae or finite automata is as hard as factoring. Technical Report TR-14-88, Harvard University Aiken Computation Laboratory, August 1988.

[11] Robert E. Schapire. The strength of weak learnability. Machine Learning, 5(2):197–227, 1990.

[12] Yoav Freund. Boosting a weak learning algorithm by majority. Information and Computation, 121(2):256–285, 1995.

[13] Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. International Journal of Pattern Recognition and Artificial Intelligence, 7(4):705– 719, 1993.

[14] Bias-Variance Decomposition - mlxtend