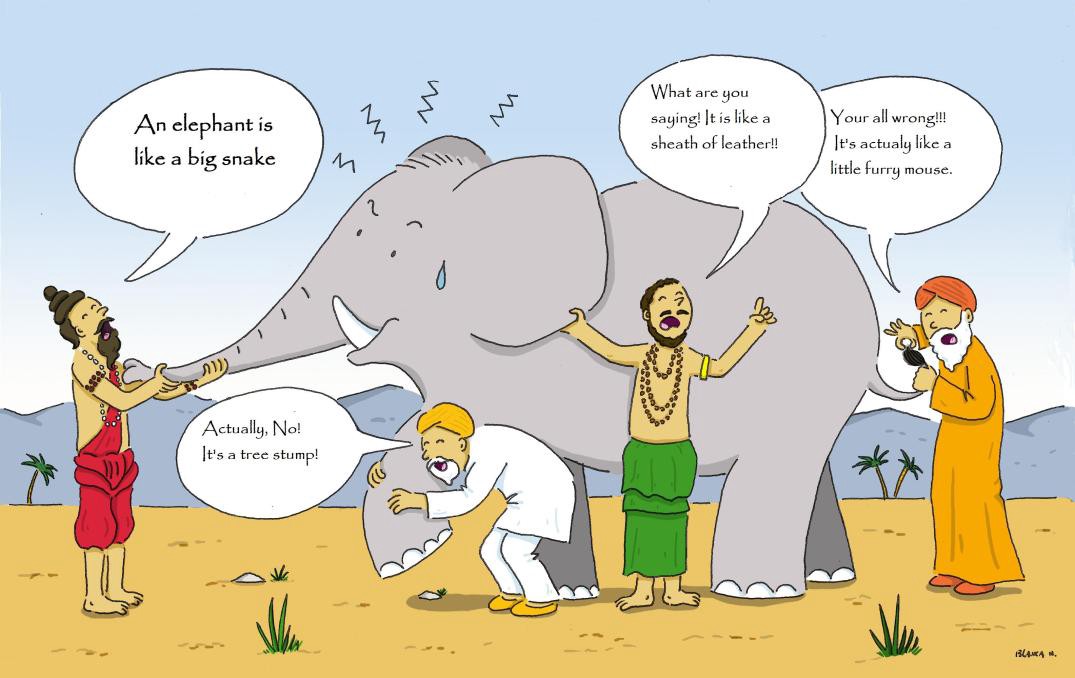
**Bagging and Boosting**

Bagging and Boosting are similar in that they are both ***ensemble techniques***, where a set of weak learners are combined to create a strong learner that obtains better performance than a single one.

**ENSEMBLE LEARNING**

Ensemble methods combine several decision trees classifiers to produce better predictive performance than a single decision tree classifier. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner, thus increasing the accuracy of the model. When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are **noise, variance, and bias**. Ensemble helps to reduce these factors (except noise, which is irreducible error).

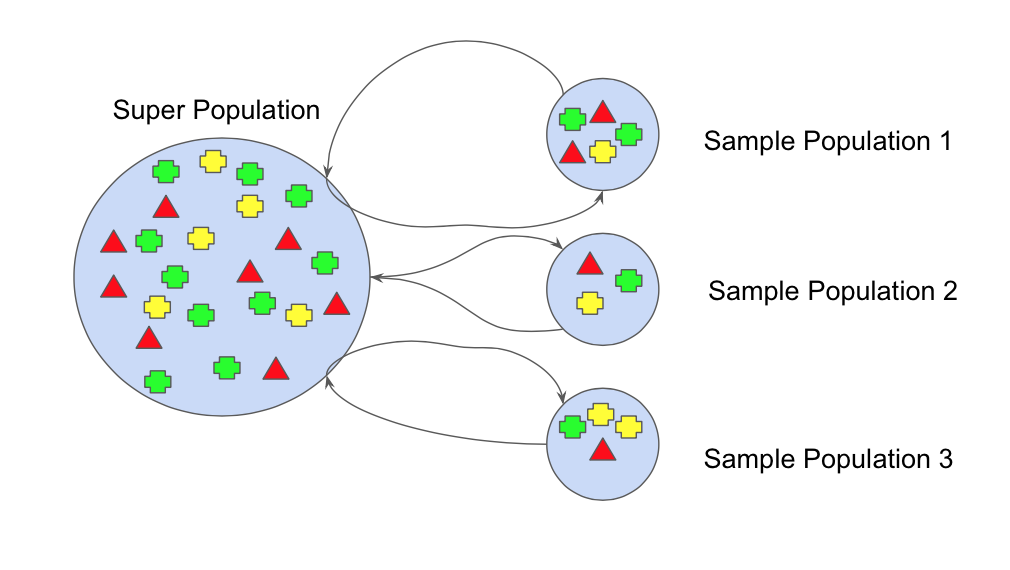


Another way to think about Ensemble learning is ***Fable of blind men and elephant.*** All of the blind men had their own description of the elephant. Even though each of the description was true, it would have been better to come together and discuss their undertanding before coming to final conclusion. This story perfectly describes the Ensemble learning method.

Using techniques like Bagging and Boosting **helps to decrease the variance** and increased the robustness of the model. Combinations of multiple classifiers decrease variance, especially in the case of unstable classifiers, and may produce a more reliable classification than a single classifier.

But before understanding Bagging and Boosting and how different classifiers are selected in the two algorithms, lets first talk about **Bootstrapping**.

**BOOTSTRAPPING**



Bootstrap refers to **random sampling with replacement**. Bootstrap allows us to better understand the bias and the variance with the dataset. Bootstrap involves random sampling of small subset of data from the dataset. This subset can be replaced. The selection of any example in the dataset has equal probability. This method can help to better understand the mean and standard deviation from the dataset.

Let’s assume we have a sample of ‘n’ values (x) and we’d like to get an estimate of the mean of the sample.

mean(x) = 1/n \* sum(x)

We know that our sample is small and that our mean has error in it. We can improve the estimate of our mean using the bootstrap procedure:

1. Create many (e.g. m) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times).
2. Calculate the mean of each sub-sample.
3. Calculate the average of all of our collected means and use that as our estimated mean for the data.

For example, let’s say we used 3 resamples and got the mean values 2.5, 3.3 and 4.7. Taking the average of these we could take the estimated mean of the data to be 3.5.

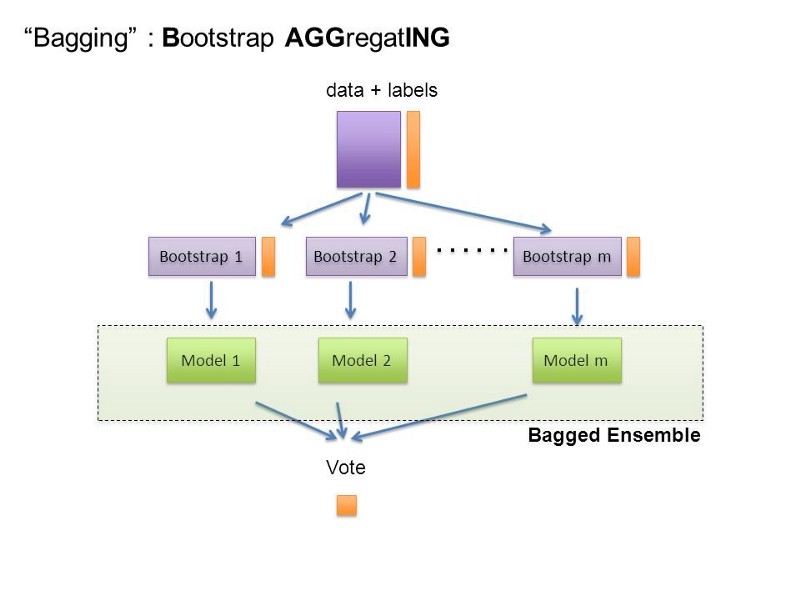
Having understood Bootstrapping we will use this knowledge to understand **Bagging** and **Boosting**.

**BAGGING**

Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method. Bagging is the application of the **Bootstrap procedure to a high-variance machine learning algorithm**, typically decision trees.

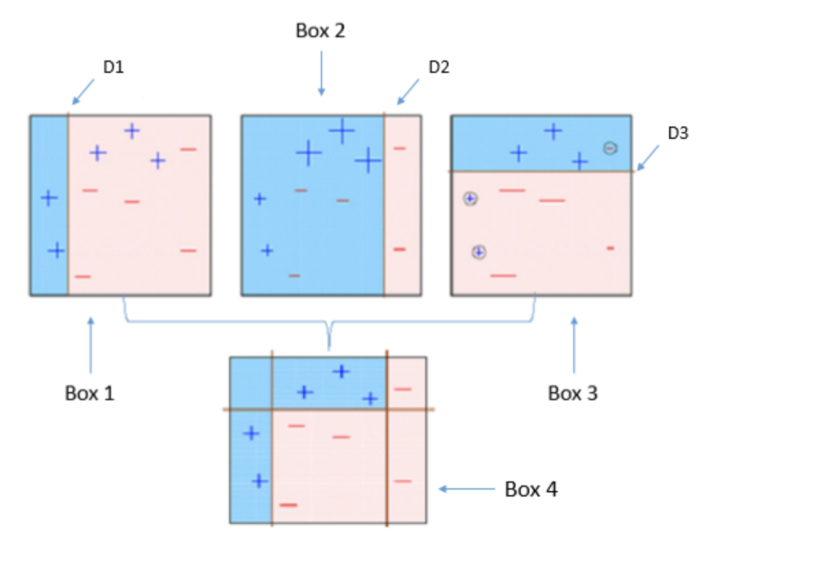
1. Suppose there are N observations and M features. A sample from observation is selected randomly with replacement (Bootstrapping).
2. A subset of features is selected to create a model with sample of observations and subset of features.
3. Feature from the subset is selected which gives the best split on the training data.(Visit my blog on Decision Tree to know more of best split)
4. This is repeated to create many models and every model is trained in parallel
5. Prediction is given based on the aggregation of predictions from all the models.

*When bagging with decision trees, we are less concerned about individual trees overfitting the training data. For this reason and for efficiency, the individual decision trees are grown deep (e.g. few training samples at each leaf-node of the tree) and the trees are not pruned. These trees will have both high variance and low bias. These are important characterize of sub-models when combining predictions using bagging. The only parameters when bagging decision trees is the number of samples and hence the number of trees to include. This can be chosen by increasing the number of trees on run after run until the accuracy begins to stop showing improvement*



**BOOSTING**

Boosting refers to a group of algorithms that utilize weighted averages to make weak learners into stronger learners. Unlike bagging that had each model run independently and then aggregate the outputs at the end without preference to any model. Boosting is all about “teamwork”. Each model that runs, dictates what features the next model will focus on.



*Box 1:* You can see that we have assigned equal weights to each data point and applied a decision stump to classify them as + (plus) or — (minus). The decision stump (D1) has generated vertical line at left side to classify the data points. We see that, this vertical line has incorrectly predicted three + (plus) as — (minus). In such case**, we’ll assign higher weights to these three + (plus) and apply another decision stump.**

*Box 2:* Here, you can see that the size of three incorrectly predicted + (plus) is bigger as compared to rest of the data points. In this case, the second decision stump (D2) will try to predict them correctly. Now, a vertical line (D2) at right side of this box has classified three mis-classified + (plus) correctly. But again, it has caused mis-classification errors. This time with three -(minus). **Again, we will assign higher weight to three — (minus) and apply another decision stump**.

*Box 3:* Here, three — (minus) are given higher weights. A decision stump (D3) is applied to predict these mis-classified observation correctly. **This time a horizontal line is generated to classify + (plus) and — (minus) based on higher weight of mis-classified observation**.

*Box 4:* Here, we have combined D1, D2 and D3 to form a strong prediction having complex rule as compared to individual weak learner. You can see that this algorithm has classified these observation quite well as compared to any of individual weak learner.

**Which is the best, Bagging or Boosting?**

* There’s not an outright winner; it depends on the data, the simulation and the circumstances.  
  Bagging and Boosting decrease the variance of your single estimate as they combine several estimates from different models. So the result may be a model with **higher stability**.
* If the problem is that the single model gets a very low performance, Bagging will rarely get a **better bias**. However, Boosting could generate a combined model with lower errors as it optimises the advantages and reduces pitfalls of the single model.
* By contrast, if the difficulty of the single model is **over-fitting**, then Bagging is the best option. Boosting for its part doesn’t help to avoid over-fitting; in fact, this technique is faced with this problem itself. For this reason, Bagging is effective more often than Boosting.

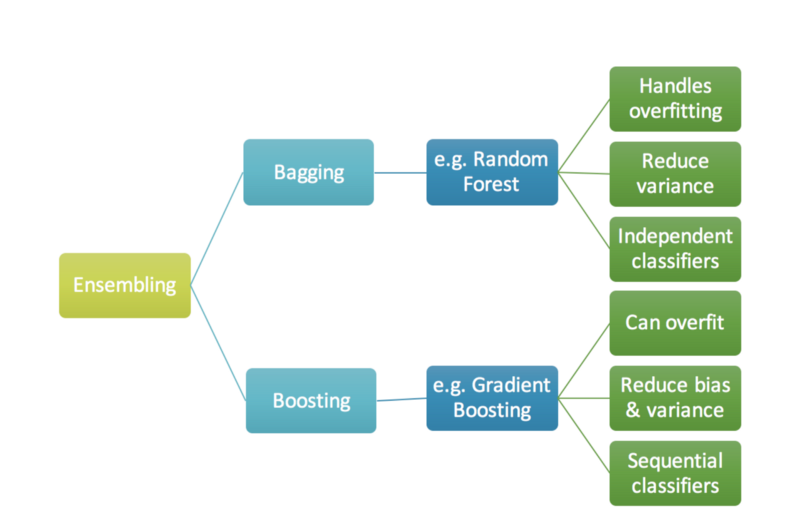
**BOOSTING**

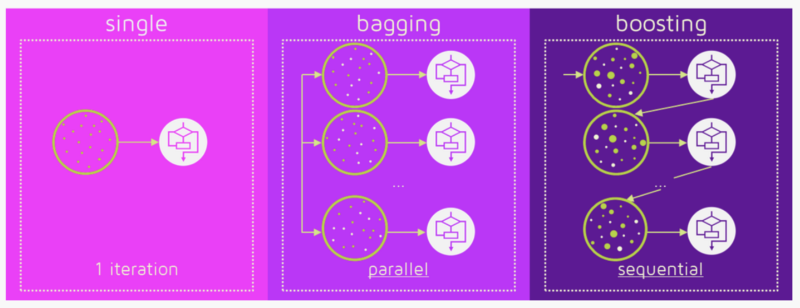
**Boosting** is an ensemble technique in which the predictors are not made independently, but sequentially.

* The main idea of boosting is **to add additional models to the overall ensemble** model **sequentially**. Previously with bagging, we averaged each individual model created.
* **This time with each iteration of boosting, a new model is created** and the new base-learner model is trained (updated) from the errors of the previous learners.

The algorithm creates multiple weak models whose output is added together to get an overall prediction.

* Unlike the bagging examples above, **classical boosting the subset creation is not random and performance will depend upon the performance of previous models.**
* As, **each new subset** which is **iterated upon contains elements which could have been misclassified by previous models**. We will also be using the same hard voting we used previously to ensemble the models together.
* This technique employs the logic in which the **subsequent predictors learn from the mistakes of the previous predictors.** Therefore, the observations have **an unequal probability** of appearing in subsequent models and ones with the highest error appear most. (So the observations are not chosen based on the bootstrap process, but based on the error).
* The predictors can be chosen from a range of models like decision trees, regressors, classifiers etc. Because new predictors are learning from mistakes committed by previous predictors, **it takes less time/iterations to reach close to actual predictions**. But we have to choose the stopping criteria carefully or it could lead to overfitting on training data.

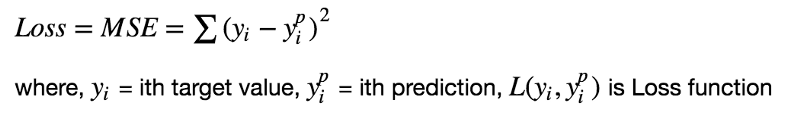




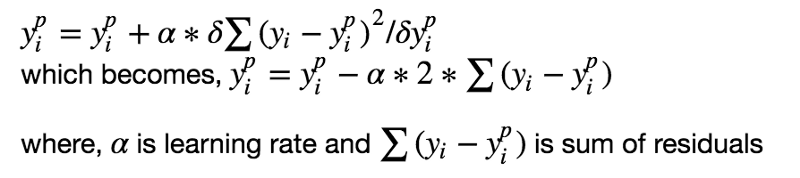
#### ****Gradient Boosting algorithm****

Gradient boosting is a machine learning technique **for regression and classification problems**, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

The objective of any supervised learning algorithm is to define a loss function and minimize it. Let’s see how math works out for Gradient Boosting algorithm. Say we have mean squared error (MSE) as loss defined as:



We want our prediction, such that our loss function (MSE) is minimum. By using **gradient descent** and updating our predictions based on a learning rate, we can find the values where MSE is minimum.

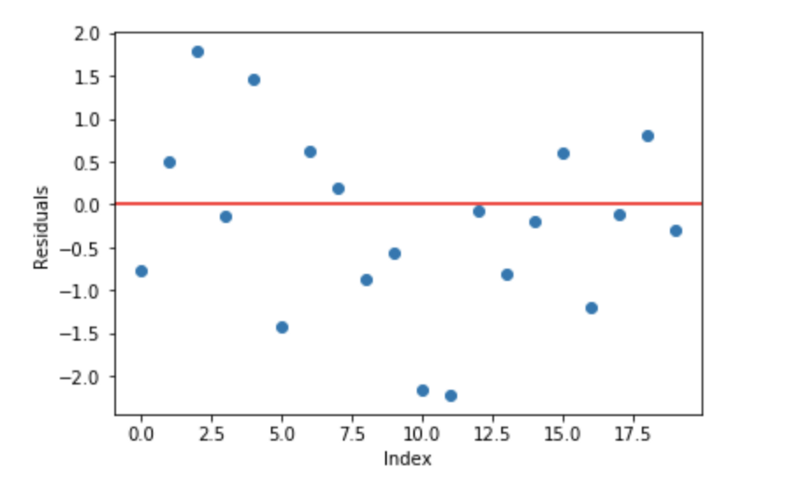


So, we are basically updating the predictions such that the sum of our residuals is close to 0 (or minimum) and predicted values are sufficiently close to actual values.

#### ****Intuition behind Gradient Boosting****

The logic behind gradient boosting is simple, (can be understood intuitively, without using mathematical notation).

A basic assumption of linear regression is that sum of its residuals is 0, i.e. the residuals should be spread randomly around zero.



**Fig 3.** Sample random normally distributed residuals with mean around 0

Now think of these residuals as mistakes committed by our predictor model. Although, tree-based models (considering decision tree as base models for our gradient boosting here) are not based on such assumptions, but if we think logically (not statistically) about this assumption, **we might argue that, if we are able to see some pattern of residuals around 0, we can leverage that pattern to fit a model.**

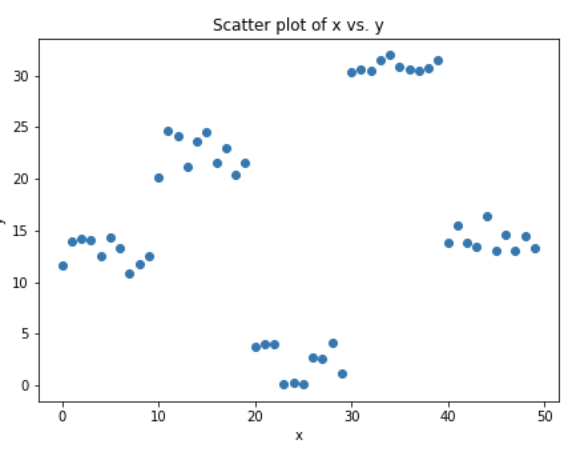
So, the intuition behind gradient boosting algorithm is to repetitively leverage the patterns in residuals and strengthen a model with weak predictions and make it better. Once we reach a stage that residuals do not have any pattern that could be modeled, we can stop modeling residuals (otherwise it might lead to overfitting). Algorithmically, we are minimizing our loss function, such that test loss reach its minima.

In summary,   
• We first model data with simple models and analyze data for errors.   
• These errors signify data points that are difficult to fit by a simple model.   
• Then for later models, we particularly focus on those hard to fit data to get them right.   
• In the end, we combine all the predictors by giving some weights to each predictor.

“The idea is to use the weak learning method several times to get a succession of hypotheses, each one refocused on the examples that the previous ones found difficult and misclassified. … Note, however, it is not obvious at all how this can be done”

#### ****Steps to fit a Gradient Boosting model****

Let’s consider simulated data as shown in scatter plot below with 1 input (x) and 1 output (y) variables.

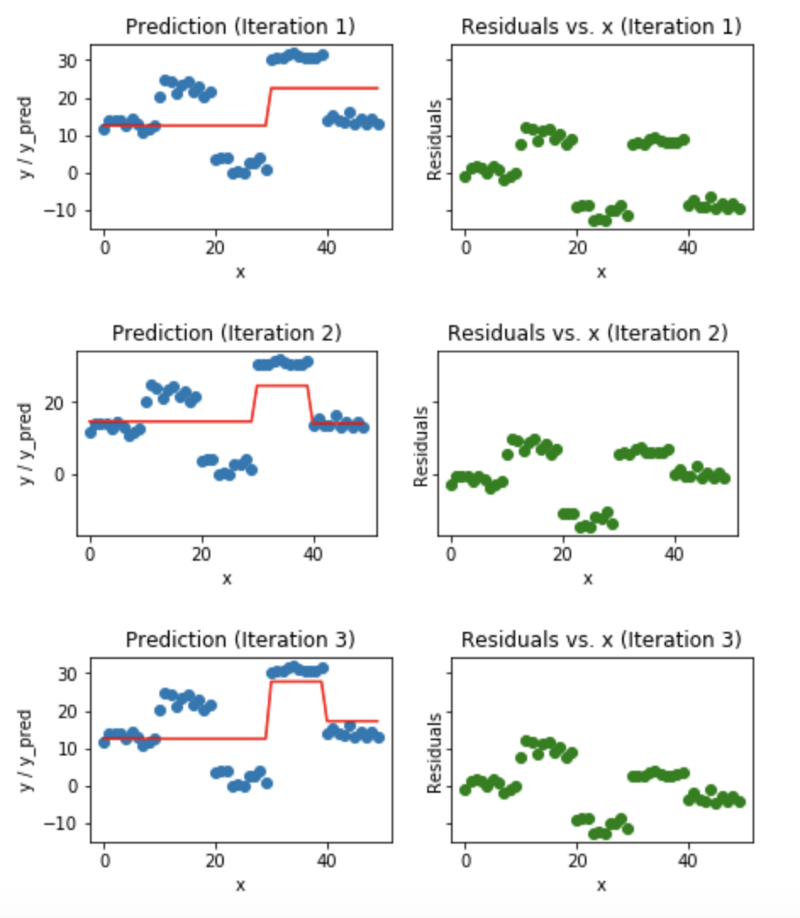


**Fig 4.** Simulated data (x: input, y: output)

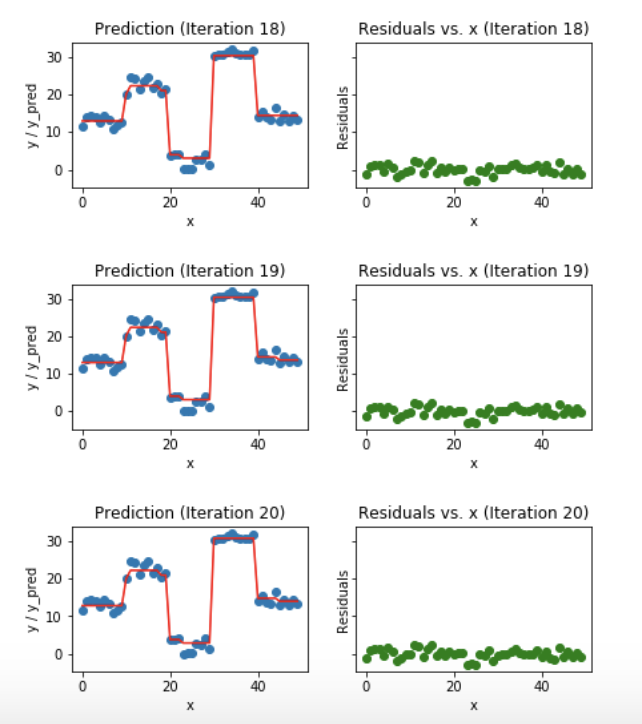
1. Fit a simple linear regressor or decision tree on data (I have chosen decision tree in my code) **[call x as input and y as output]**
2. Calculate error residuals. Actual target value, minus predicted target value **[e1= y - y\_predicted1 ]**
3. Fit a new model on error residuals as target variable with same input variables **[call it e1\_predicted]**
4. Add the predicted residuals to the previous predictions  
   **[y\_predicted2 = y\_predicted1 + e1\_predicted]**
5. Fit another model on residuals that is still left. i.e. **[e2 = y - y\_predicted2]** and repeat steps 2 to 5 until it starts overfitting or the sum of residuals become constant. Overfitting can be controlled by consistently checking accuracy on validation data.

#### ****Visualization of working Gradient Boosting Tree****

Blue dots (left) plots are input (x) vs. output (y) • Red line (left) shows values predicted by decision tree • Green dots (right) shows residuals vs. input (x) for ith iteration • Iteration represent sequential order of fitting gradient boosting tree



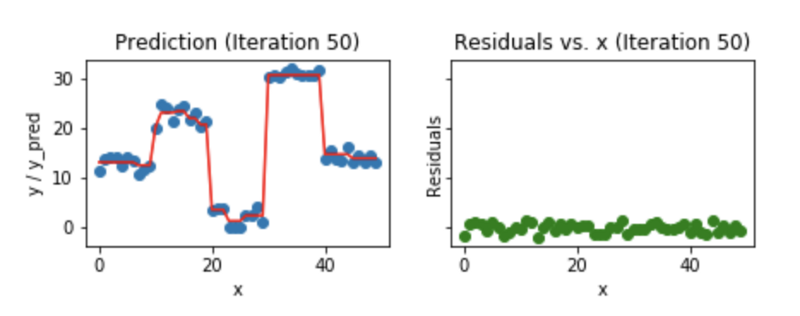
**Fig 5.** Visualization of gradient boosting predictions (First 4 iterations)



**Fig 6.** Visualization of gradient boosting predictions (18th to 20th iterations)

We observe that after 20th iteration , residuals are randomly distributed (I am not saying random normal here) around 0 and our predictions are very close to true values. (iterations are called n\_estimators in sklearn implementation). This would be a good point to stop or our model will start overfitting.

Let’s see how our model look like for 50th iteration.



**Fig 7**. Visualization of gradient boosting prediction (iteration 50th)

We see that even after 50th iteration, residuals vs. x plot look similar to what we see at 20th iteration. But the model is becoming more complex and predictions are overfitting on the training data and are trying to learn each training data. So, it would have been better to stop at 20th iteration.

## AdaBoost

**AdaBoost is best used to boost the performance of decision trees on binary classification problems.**

AdaBoost was originally called AdaBoost.M1 by the authors of the technique Freund and Schapire. More recently it may be referred to as discrete AdaBoost because it is used for classification rather than regression.

AdaBoost can be used to boost the performance of any machine learning algorithm. It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem.

The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only contain one decision for classification, they are often called decision stumps.

Each instance in the training dataset is weighted. The initial weight is set to:

weight(xi) = 1/n

Where xi is the i’th training instance and n is the number of training instances