Boosting

Introduction & Gradient Boosting

Data Science Immersive

Credit: Frank Burkholder, edited by Jessica Curley

Objectives

After this lecture you will be able to:

- Describe conceptually what boosting is
- Describe the Gradient Boosting algorithm
- List Gradient Boosting hyperparameters
- List a couple of *useful* non-sklearn based boosting algorithms

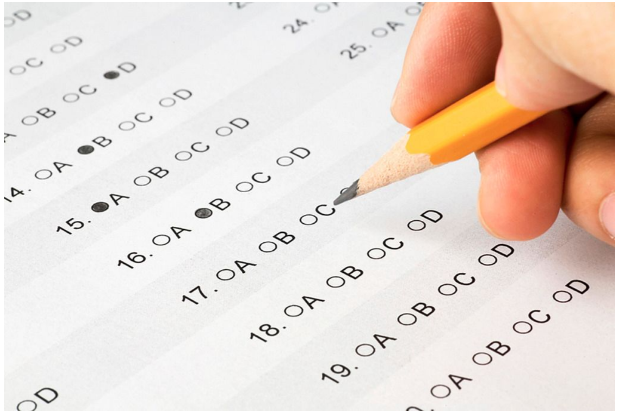
Setting the stage

Test time!

10 students have 1 hour to take the test.

Think of each student taking the test as a machine learning model (say decision tree).

After the test, and by using all the tests, is there some way the **students could work together** to submit **one exam** with the **answers mostly right**?

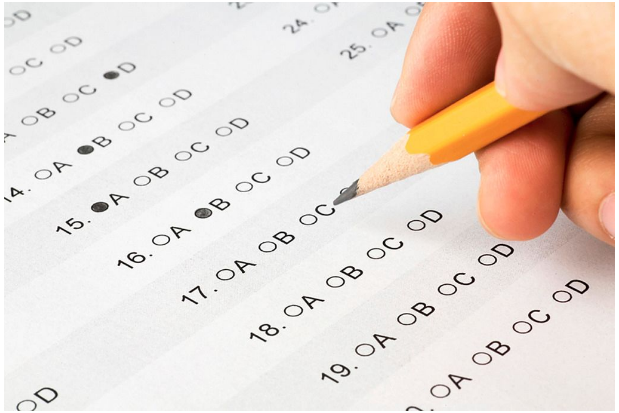


Working apart, then aggregating results...

Assuming the students are *strong learners* (low bias) but make occasional mistakes (high variance), then for each multiple choice question we take the majority vote (mode) from the class.

That majority vote should be right (and will have less variance than an answer by an individual student.)

It’s like the ensemble method **bagging** (but without the bootstrap - all students took the same test).

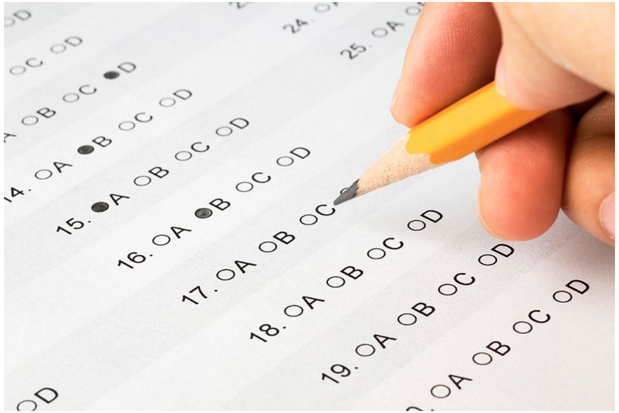


An unconventional strategy...

But what if students could work differently during the test?

Instead of working in parallel (each student working independently on their own during the hour), they could work **sequentially**?

Student 1 fills in the test quickly (only has 6 minutes!), then passes the test to Student 2 for editing, who passes it to Student 3 for editing, etc. all the way to student 10.

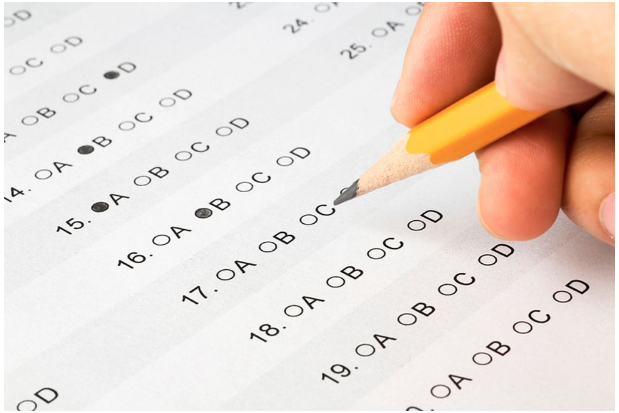


...but it’s not cheating in the machine learning world...

But what if students could work differently during the test?

Instead of working in parallel (each student working independently on their own during the hour), they could work **sequentially**?

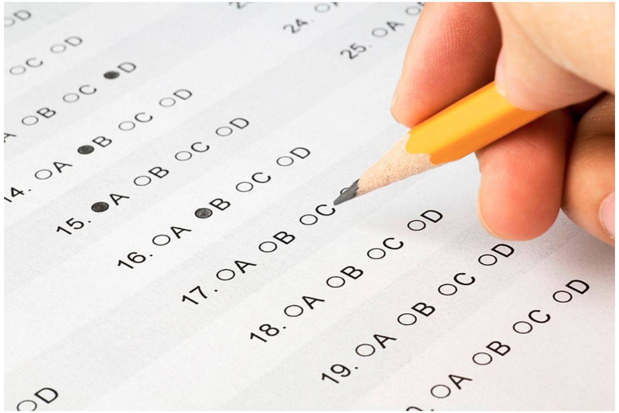
Student 1 fills in the test quickly (only has 6 minutes!), then passes the test to Student 2 for editing, who passes it to Student 3 for editing, etc. all the way to student 10.



An important detail: each time the test is passed from one student to another, the teacher rushes in with a red pen and marks all the wrong answers for the next student to see.

That test was **boosted**!

Even though each student didn't have much time with the test (*weak learner with high bias*), the ability to focus on the questions the previous student got wrong in the end gives good results!



Ensemble Methods

The goal of **ensemble methods** is to combine the predictions of several models of a given learning algorithm to improve generalizability robustness over a single model.

Two families of ensemble models:

In **averaging methods**, high variance models are built and then their predictions are averaged. The combined model is usually better than any single model because its variance is reduced. (ex. bagging, random forests)

In **boosting methods**, high bias models are built sequentially on top of each other. The motivation is to combine many weak models to produce a powerful ensemble low-bias model in the end. (ex. boosted trees)

In both cases, the goal is to arrive at a low bias, low variance model.

Boosting - General Idea

- In Boosting, an ensemble of trees is grown *sequentially*, based on information learned in the previous tree.
- Specifically, each new tree (after the first) is fit on some **modified version of the training set** based on results from the previous tree.
- In **Adaboost**, the data associated with the largest residuals are *weighted the most* in the new training set. (Adaboost is used for classification problems)
- In **Gradient Boosting**, the new training set **is the residuals**. (Makes "new" training sets)
- We think of each boosted tree as a weak "rule of thumb," where the final prediction is a combination of these increasingly specific rules-of-thumb to yield good prediction accuracy in the end.
- We want our learners to be weak learners: high bias, low variance. By sequential addition of models the bias is decreased to get a low bias, low variance model.

How is fitting on the residual gradient descent?

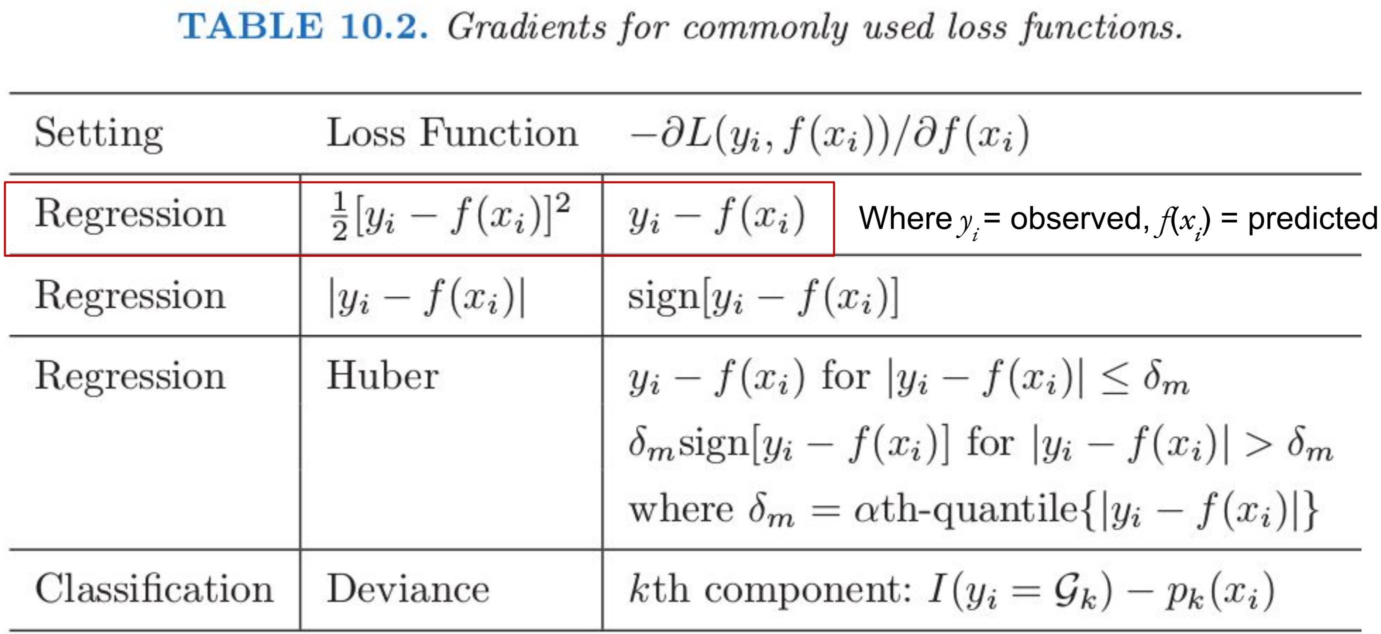


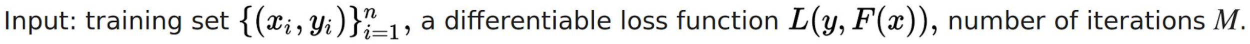
TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha\text{th-quantile}\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Where y_i = observed, $f(x_i)$ = predicted

Hastie, ESL, 2008

The Gradient Descent Regressor Algorithm



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

M - number of trees

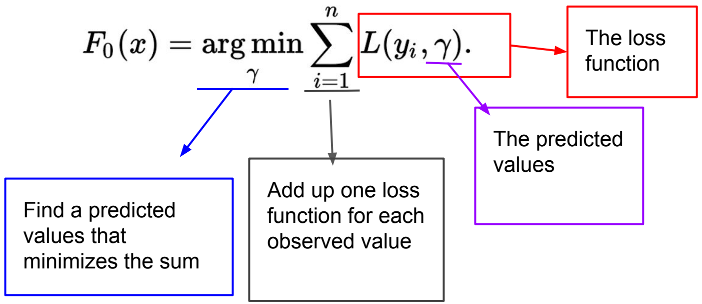
adapted from Wikipedia

The Gradient Descent Regressor Algorithm

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:



$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

The loss function

The predicted values

Find a predicted values that minimizes the sum

Add up one loss function for each observed value

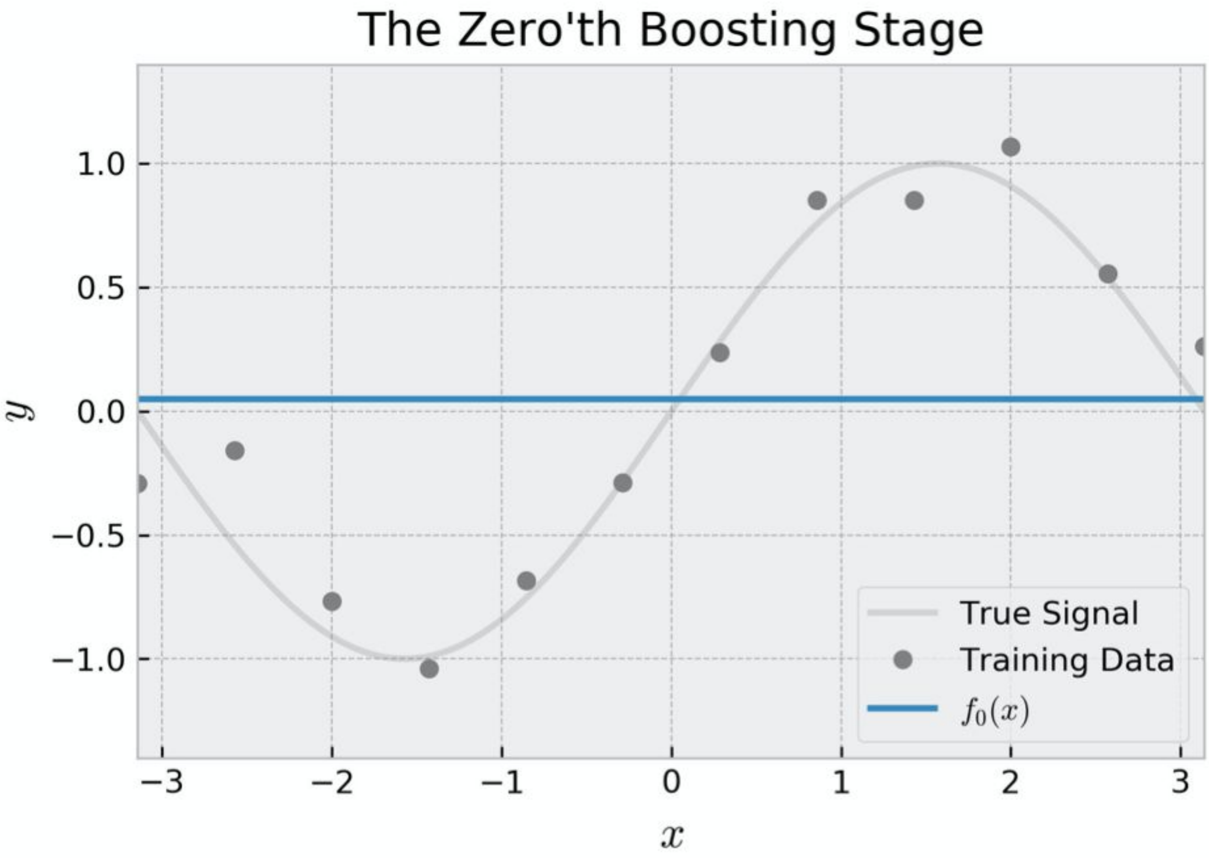
M - number of trees

Find the initial simple model with the least loss.

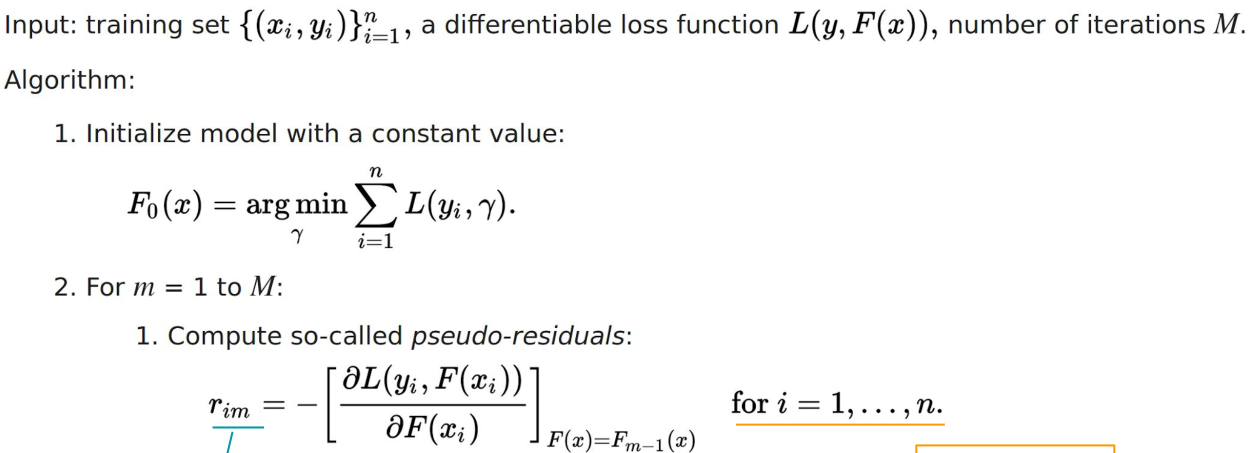
We can take the derivative of each term with respect to the predicted value, set the sum of the derivatives equal to zero and solve for the gamma that gives the minimum of the sum. ***Often this means the new predicted target is just average of the observed weights).***

adapted from Wikipedia

The Gradient Descent Regressor Visually



The Gradient Descent Regressor Algorithm



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:

    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

Pseudo-residuals:
r is the residual, *i* is the sample number and *m* is the current tree instance we're trying to build

The gradient:
Derivative of the loss function with respect to the Predicted value

Where F(x) is set to be the previous predicted value

Calculate these residuals for all *n* samples in the dataset

M - number of trees

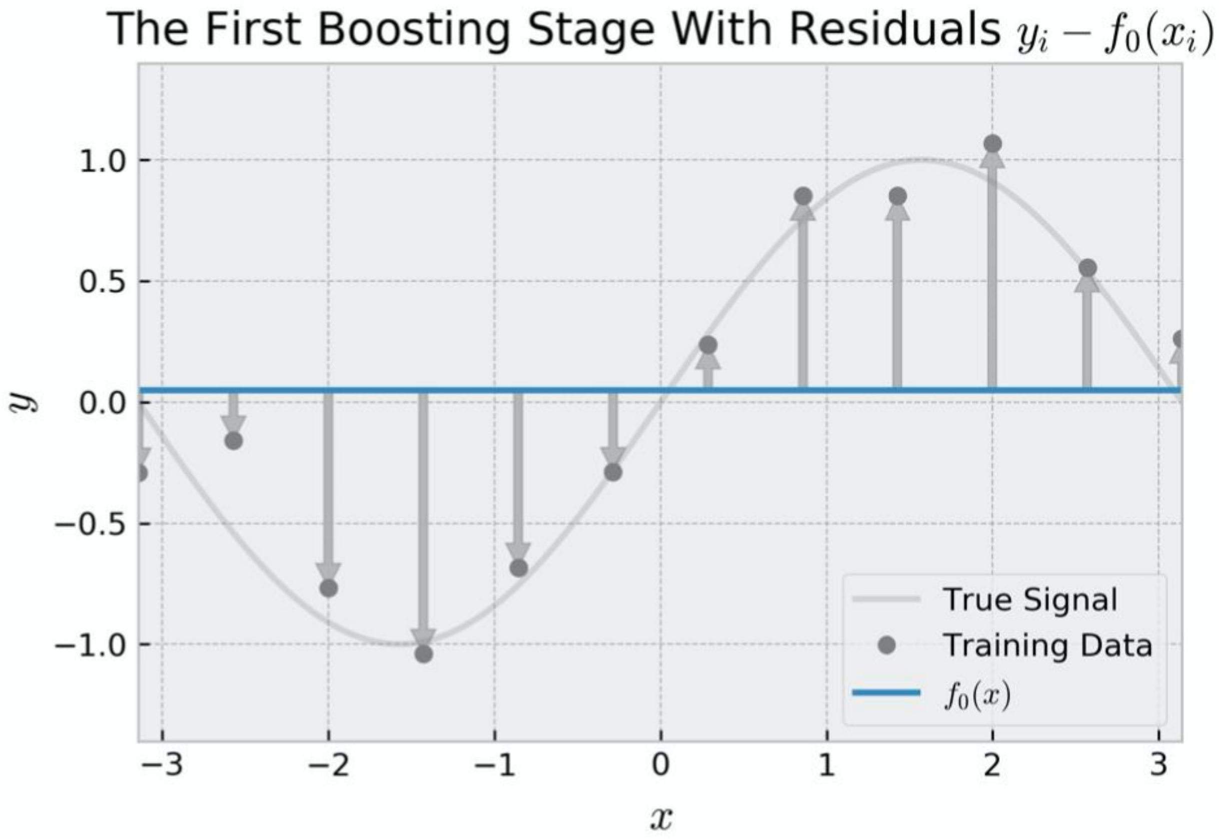
Find the initial simple model with the least loss

For each tree m = 1…M

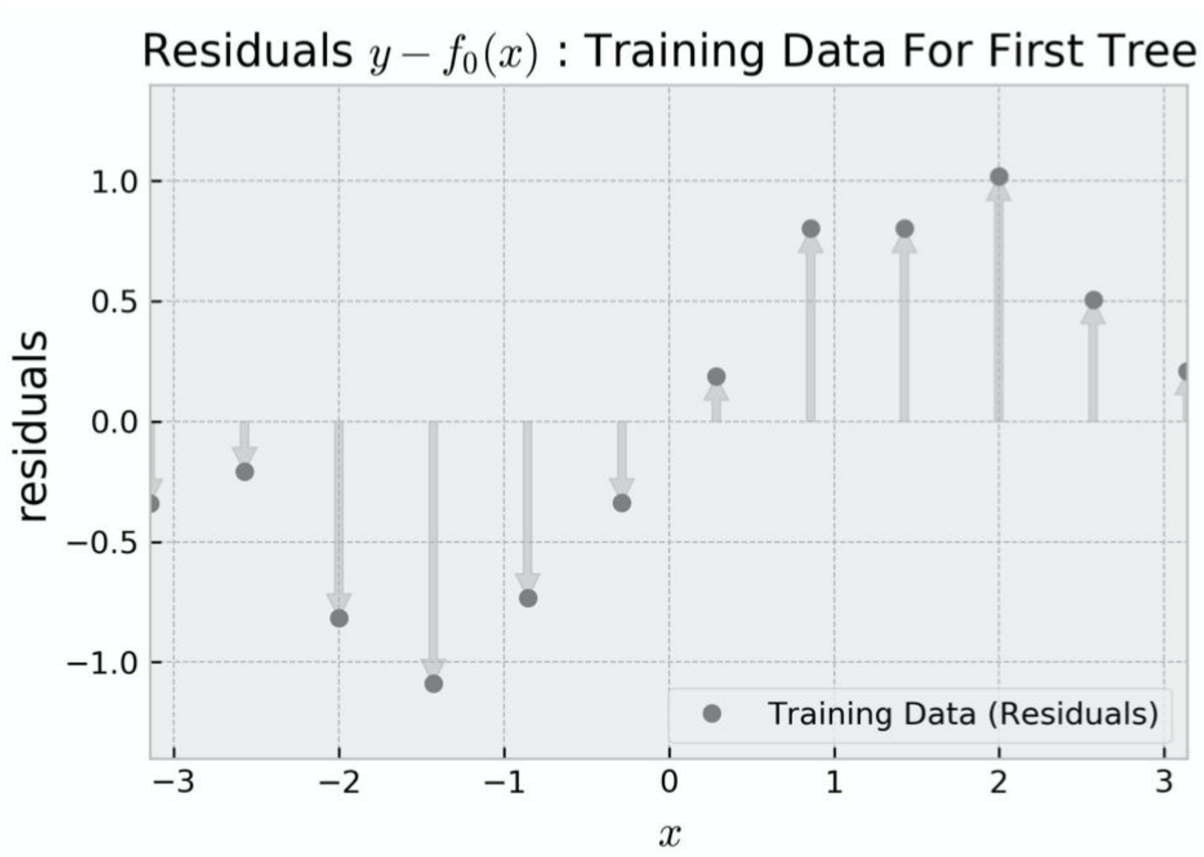
Compute the residuals for all the data i=1..n for model m-1

adapted from Wikipedia

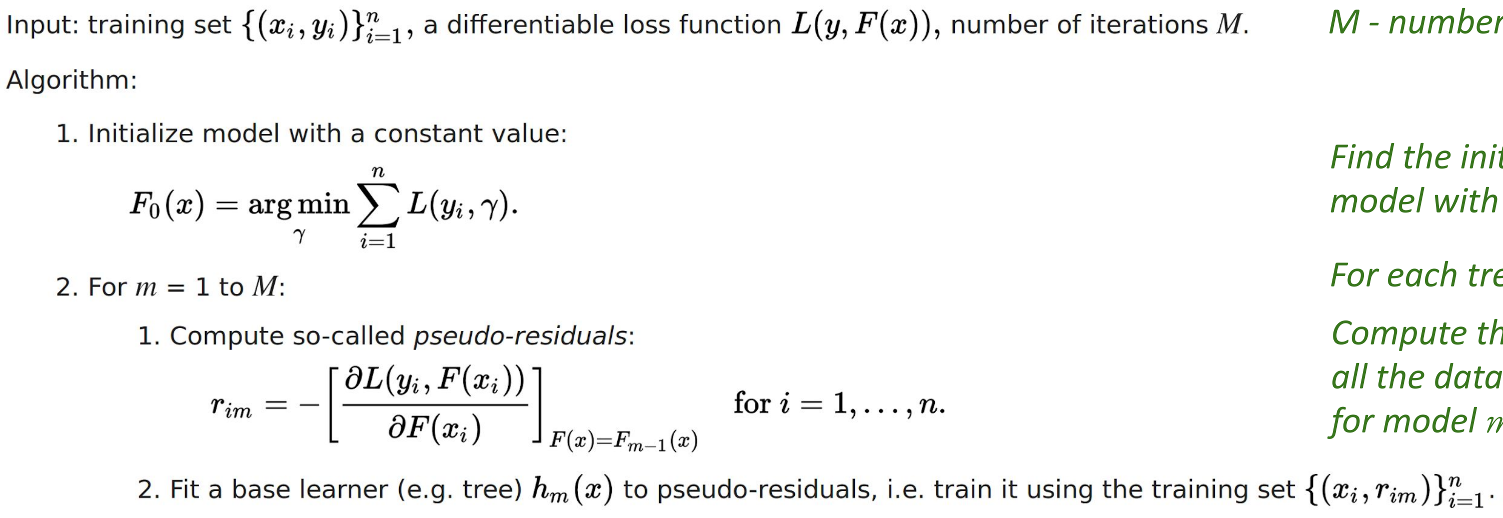
The Gradient Descent Regressor Visually



The Gradient Descent Regressor Visually



The Gradient Descent Regressor Algorithm



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.

M - number of trees

Find the initial simple model with the least loss

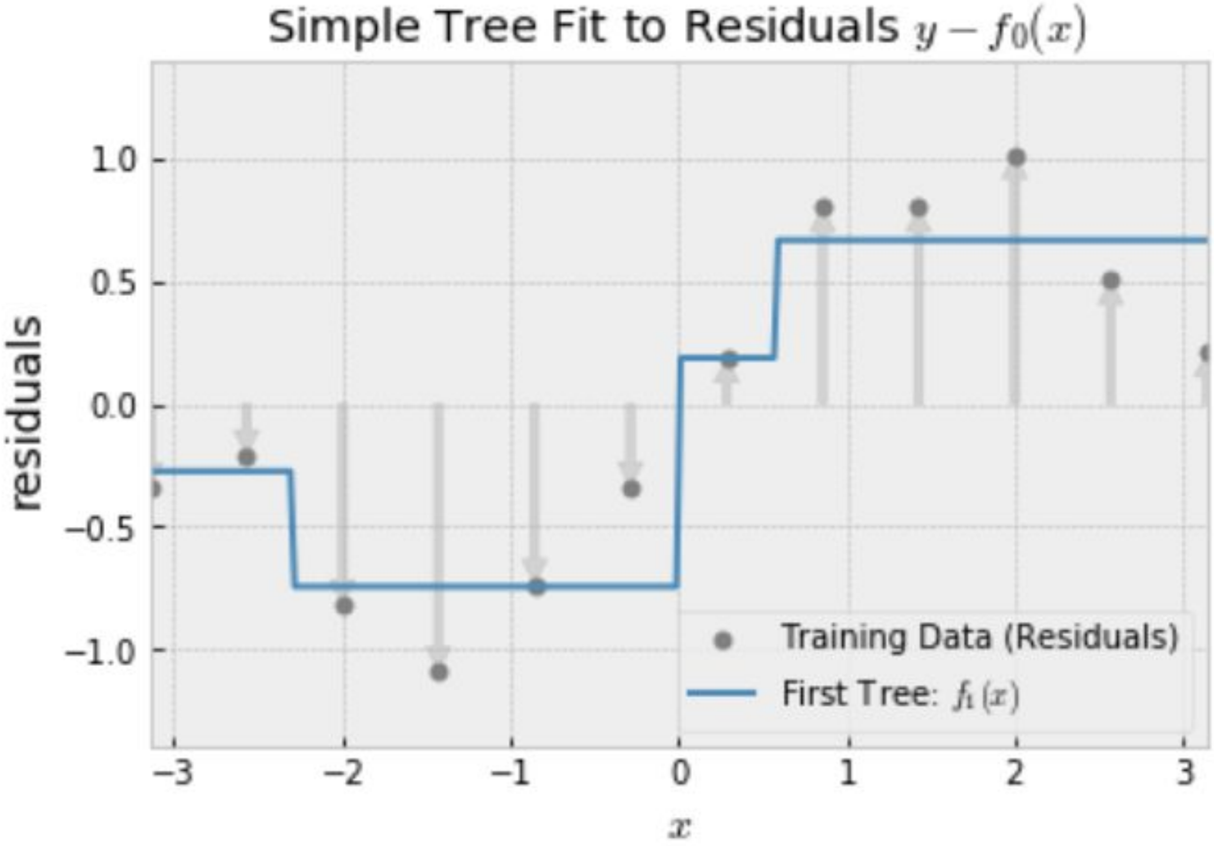
For each tree m = 1..M

Compute the residuals for all the data i=1..n for model m-1

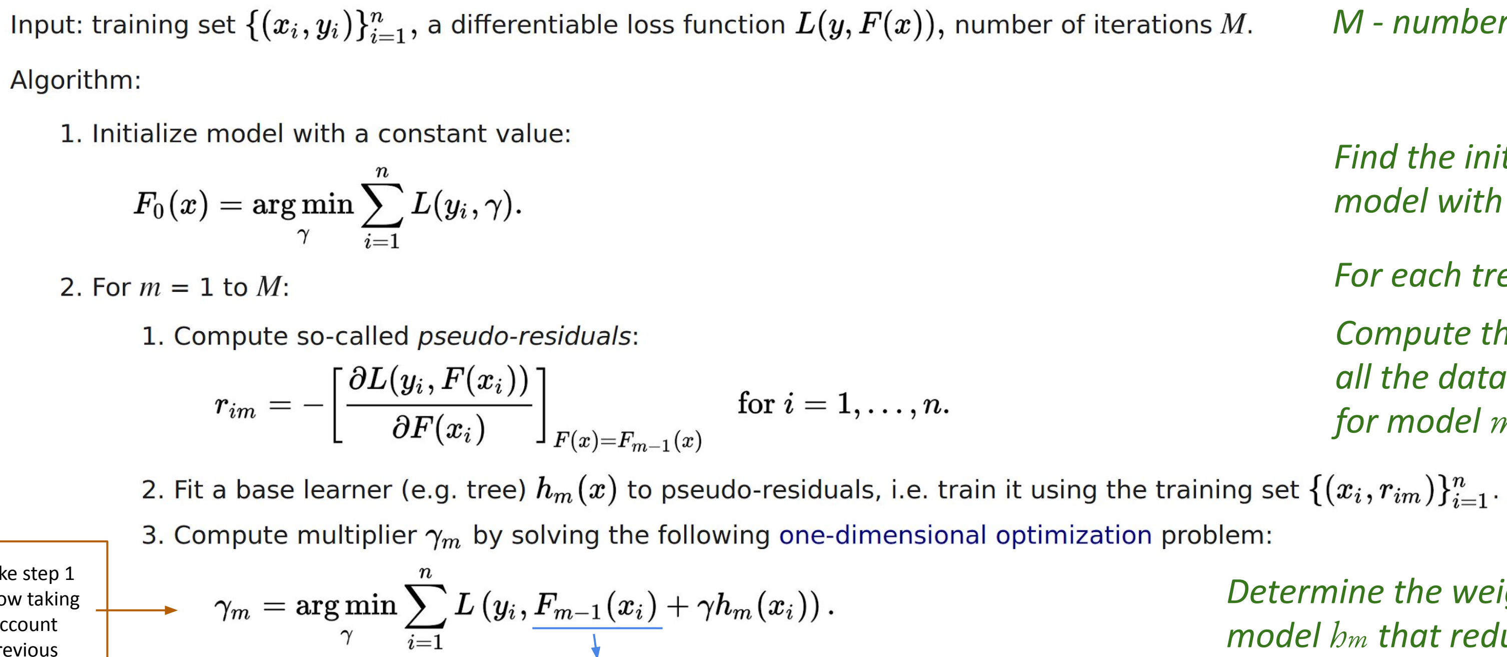
Fit a model, h_m, to residuals

adapted from Wikipedia

The Gradient Descent Regressor Visually



The Gradient Descent Regressor Algorithm



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, \underline{F_{m-1}(x_i)} + \gamma h_m(x_i)\right).$$

Just like step 1 but now taking into account the previous prediction

The most recent predicted value for x_i

M - number of trees

Find the initial simple model with the least loss

For each tree m = 1..M

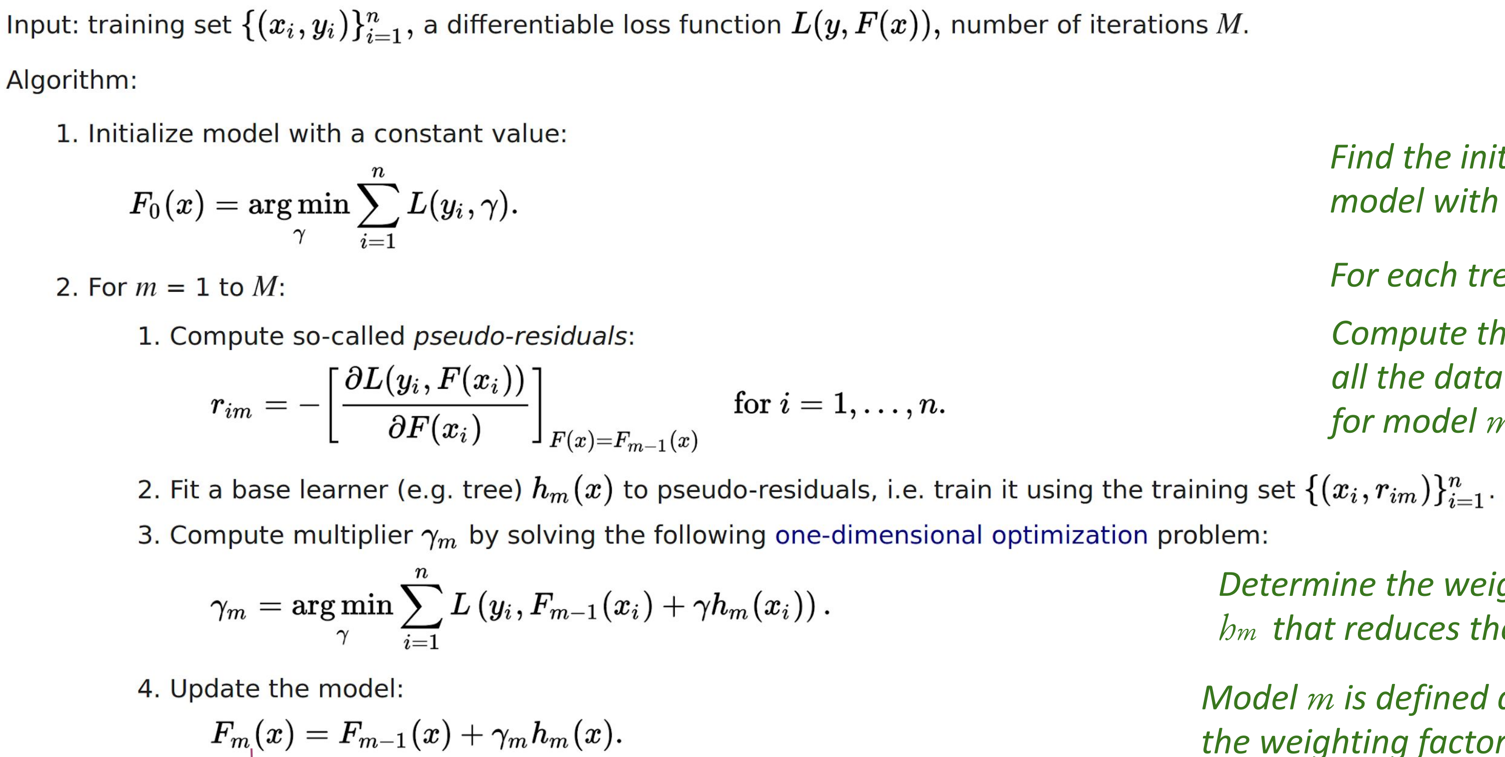
Compute the residuals for all the data i=1..n for model m-1

Fit a model, h_m, to residuals

Determine the weighting factor γ for model h_m that reduces the loss the most

adapted from Wikipedia

The Gradient Descent Regressor Algorithm



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

Make a new prediction for each sample

Find the initial simple model with the least loss

For each tree m = 1..M

Compute the residuals for all the data i=1..n for model m-1

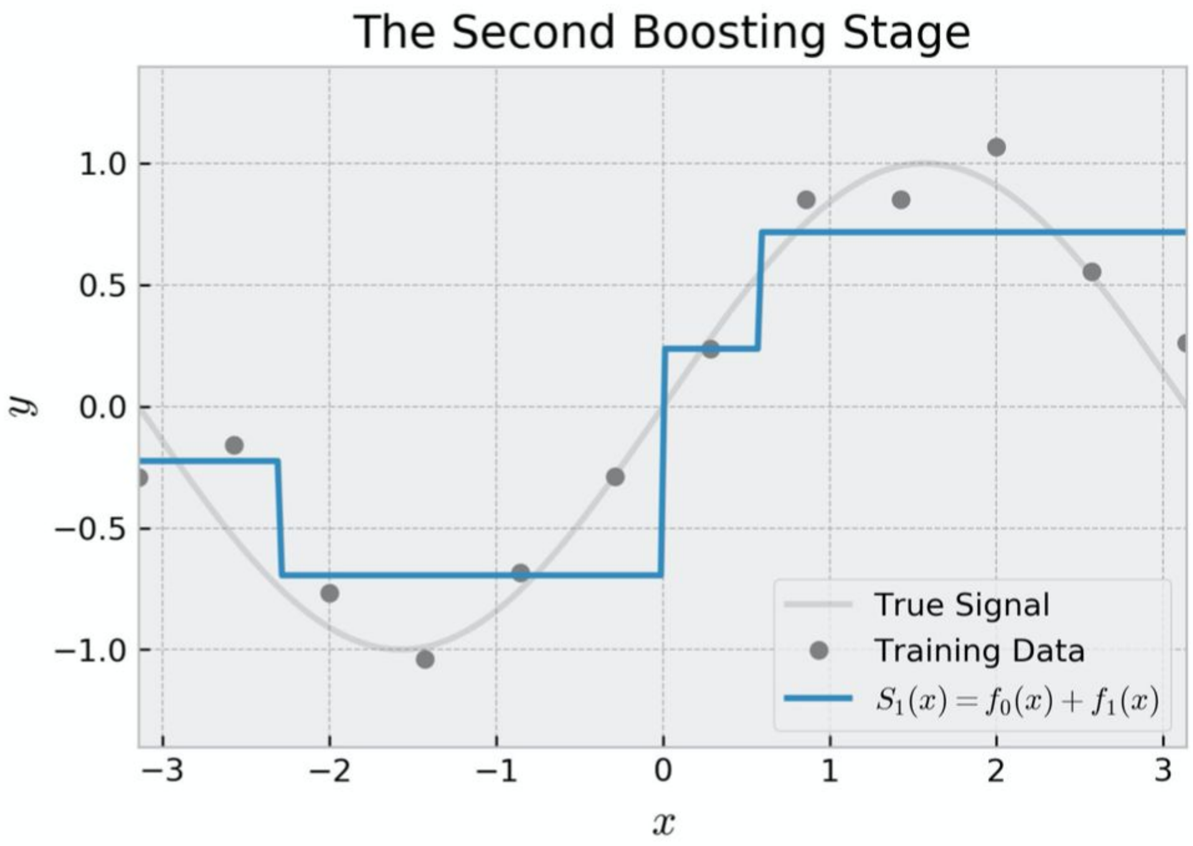
Fit a model, h_m, to residuals

Determine the weighting factor γ for h_m that reduces the loss the most

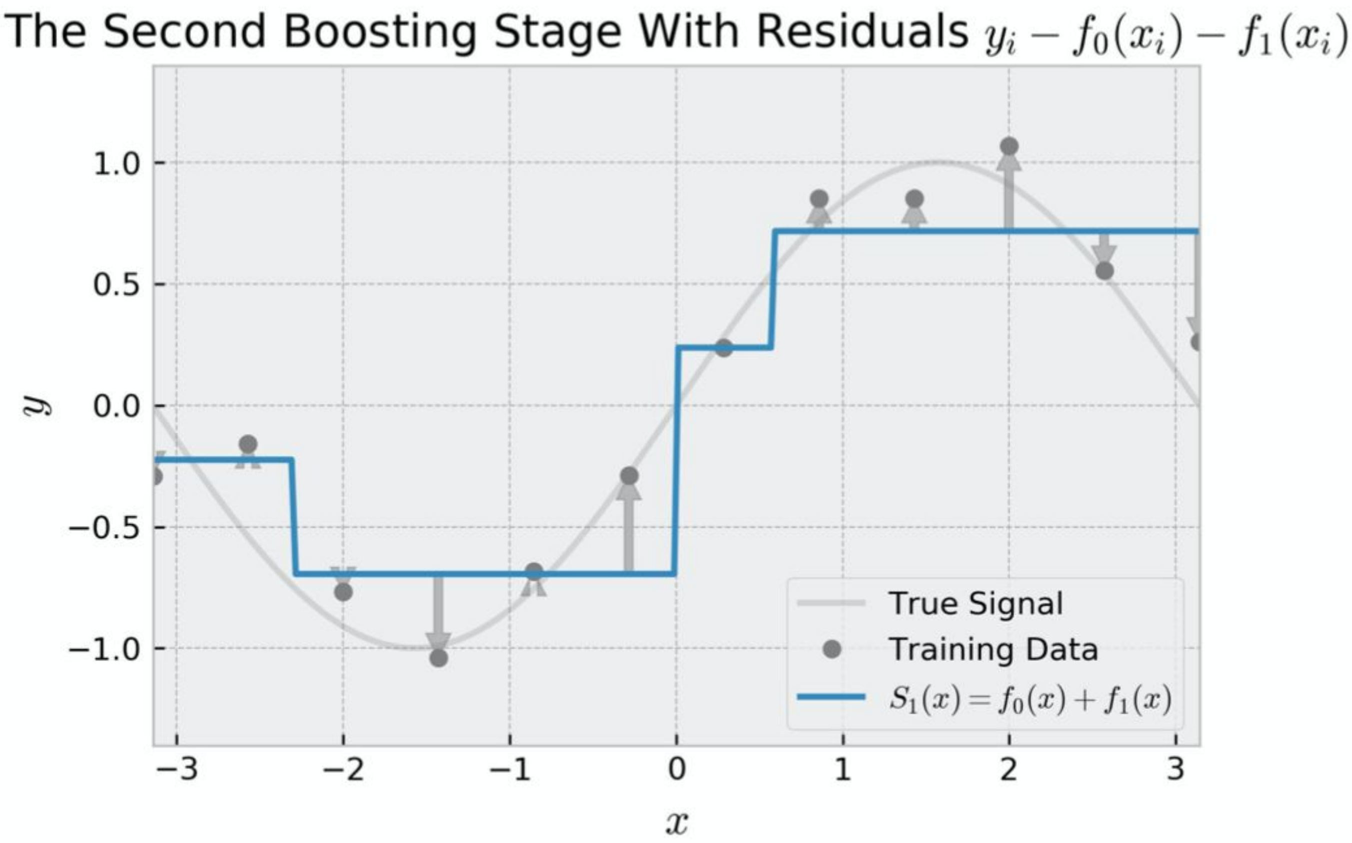
Model m is defined as model m-1 plus the weighting factor times the fit to the residuals

adapted from Wikipedia

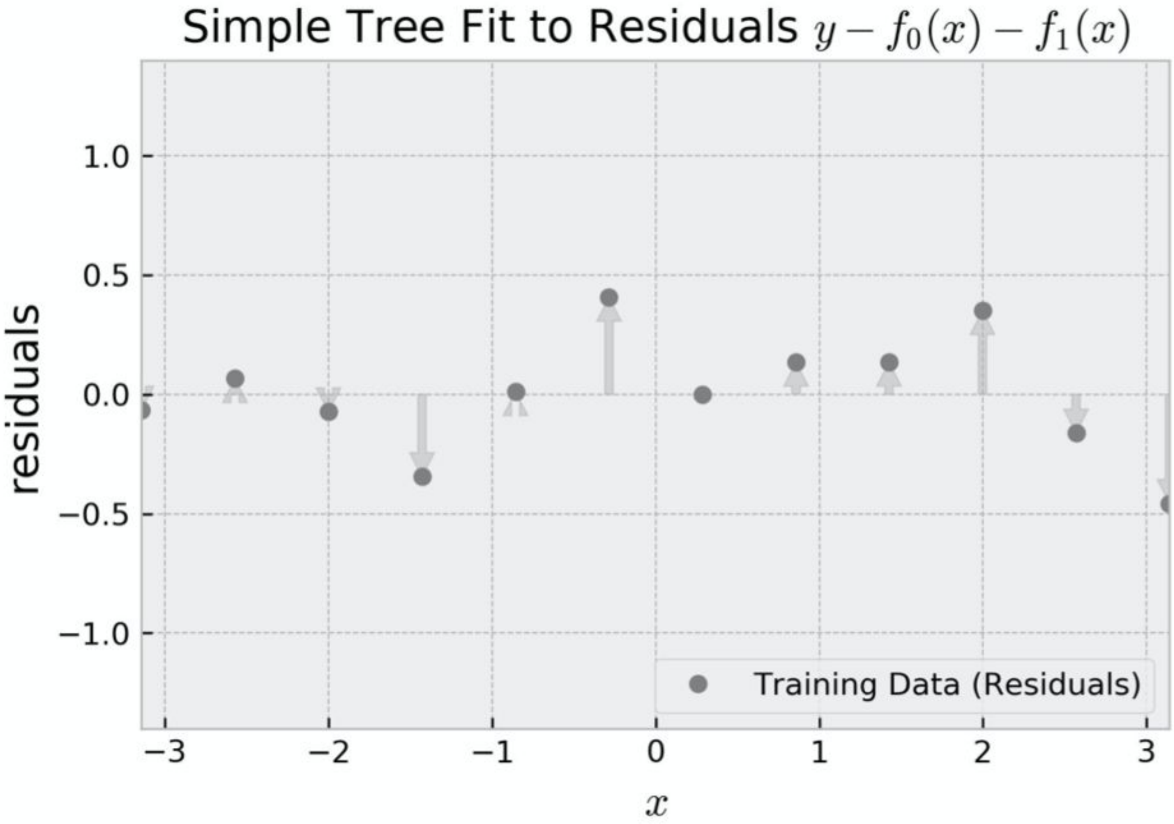
The Gradient Descent Regressor Visually



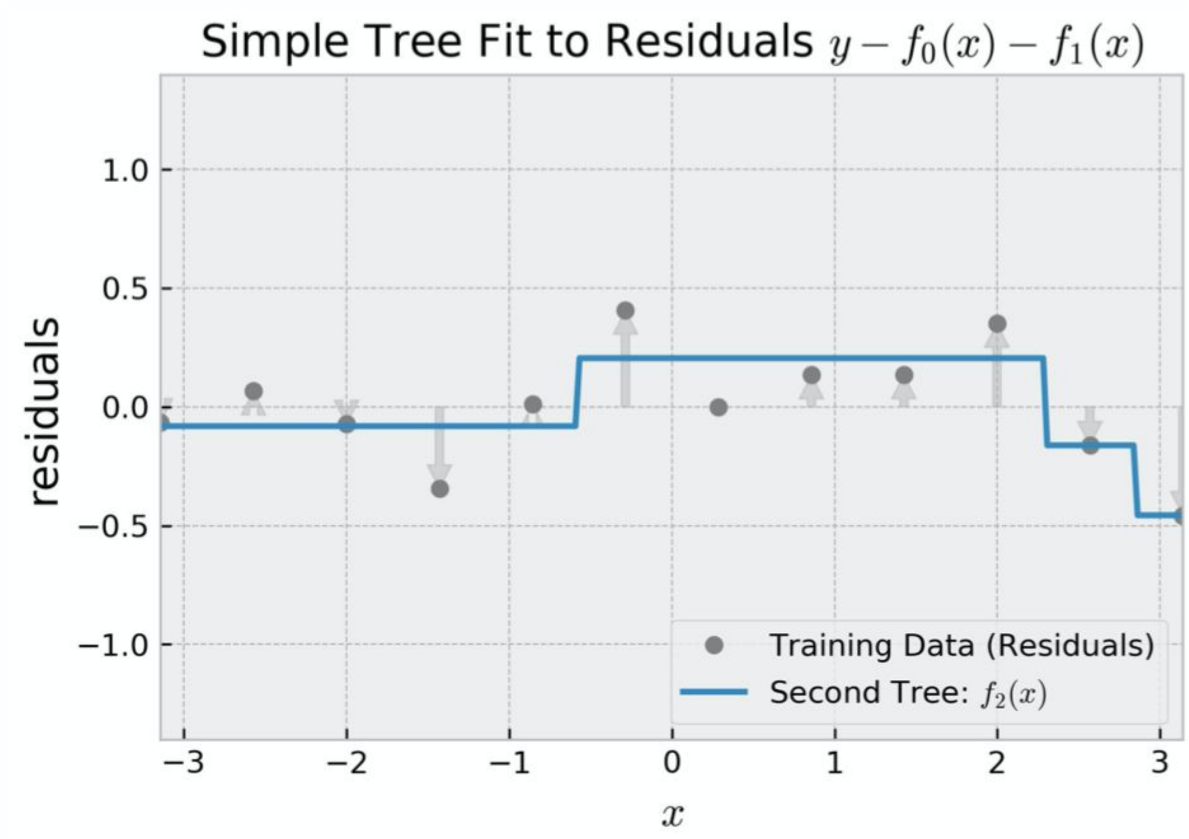
The Gradient Descent Regressor Visually



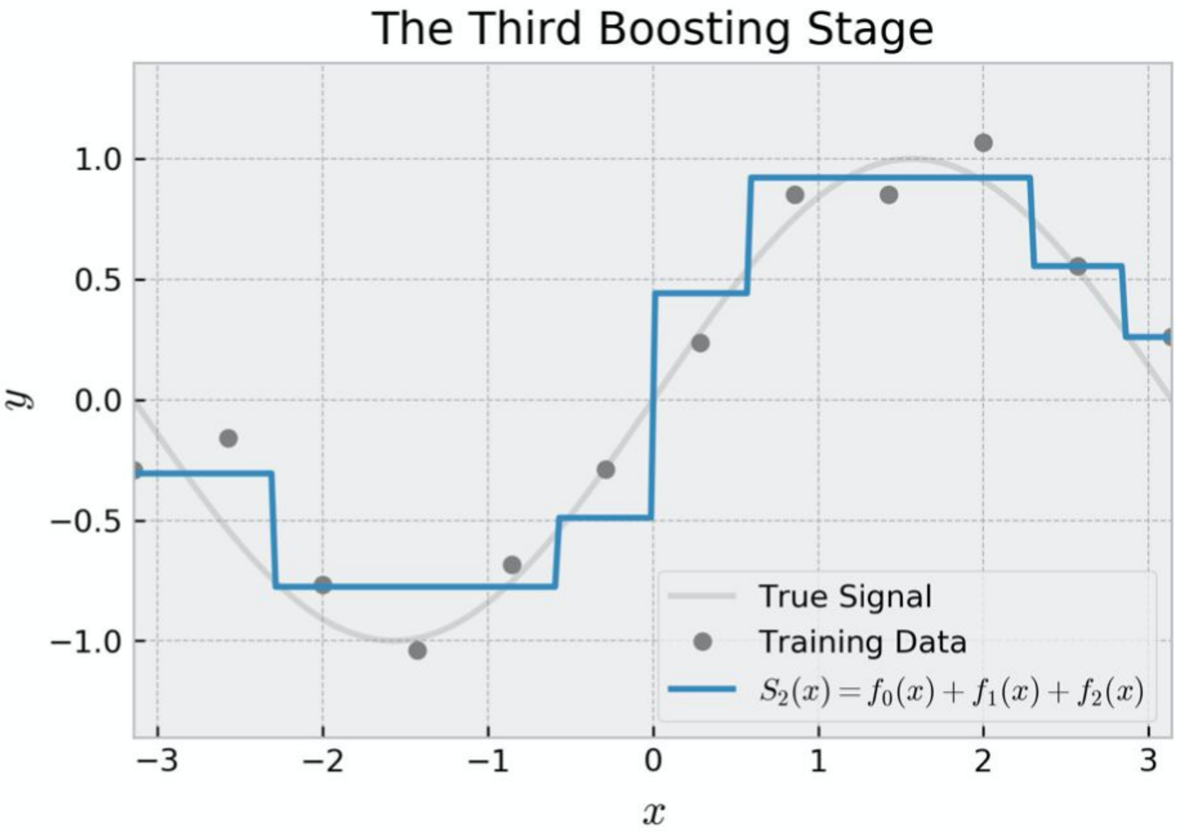
The Gradient Descent Regressor Visually



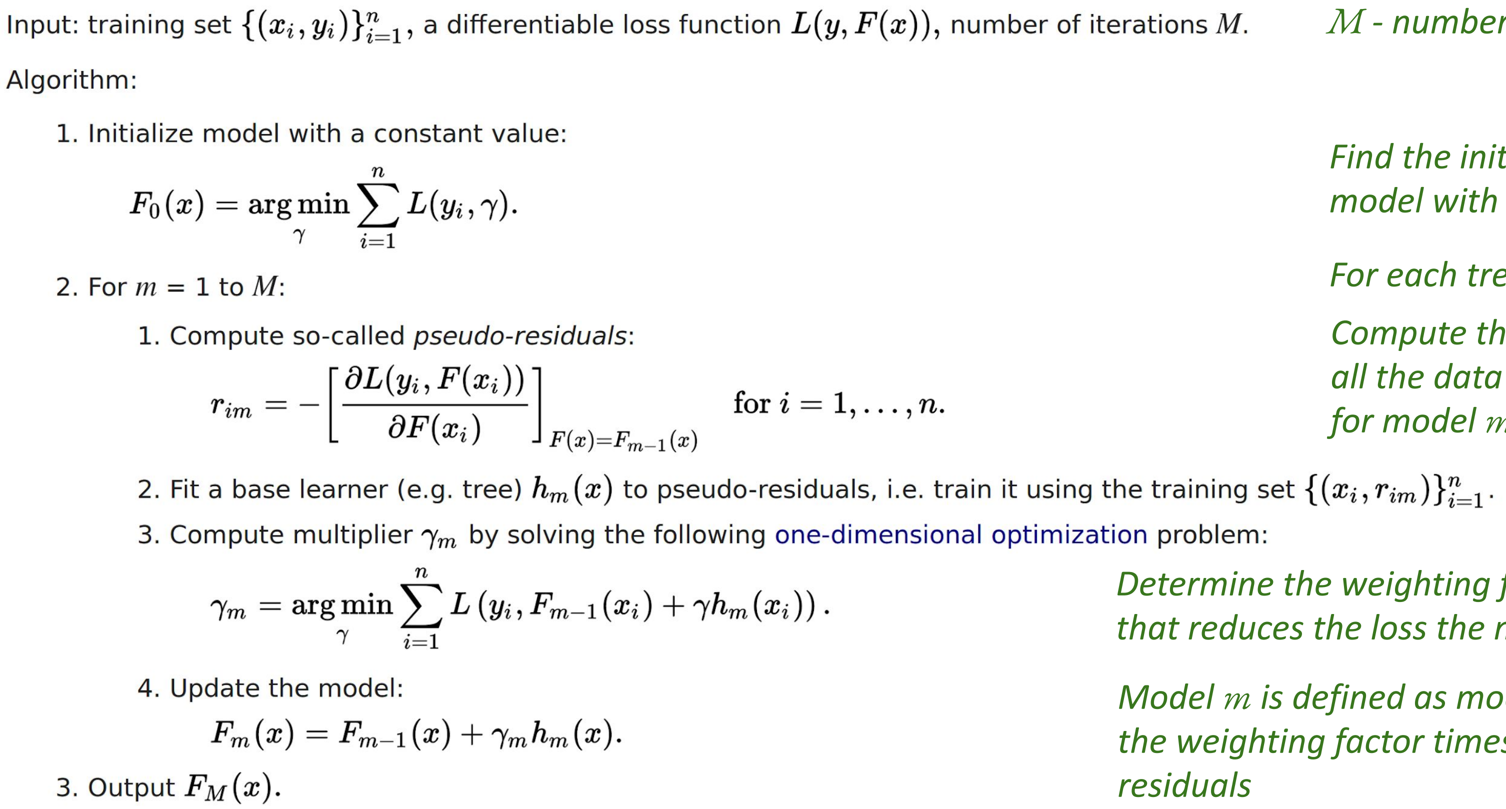
The Gradient Descent Regressor Visually



The Gradient Descent Regressor Visually



The Gradient Descent Regressor Algorithm



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:

    $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

    $$\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

    $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

M - number of trees

Find the initial simple model with the least loss

For each tree m = $1..M$

Compute the residuals for all the data i=$1..n$ for model m-1

Fit a model, h_m, to residuals

Determine the weighting factor γ for h_m that reduces the loss the most

Model m is defined as model m-1 plus the weighting factor times the fit to the residuals

Model M is the weighted sum of models 1..M-1

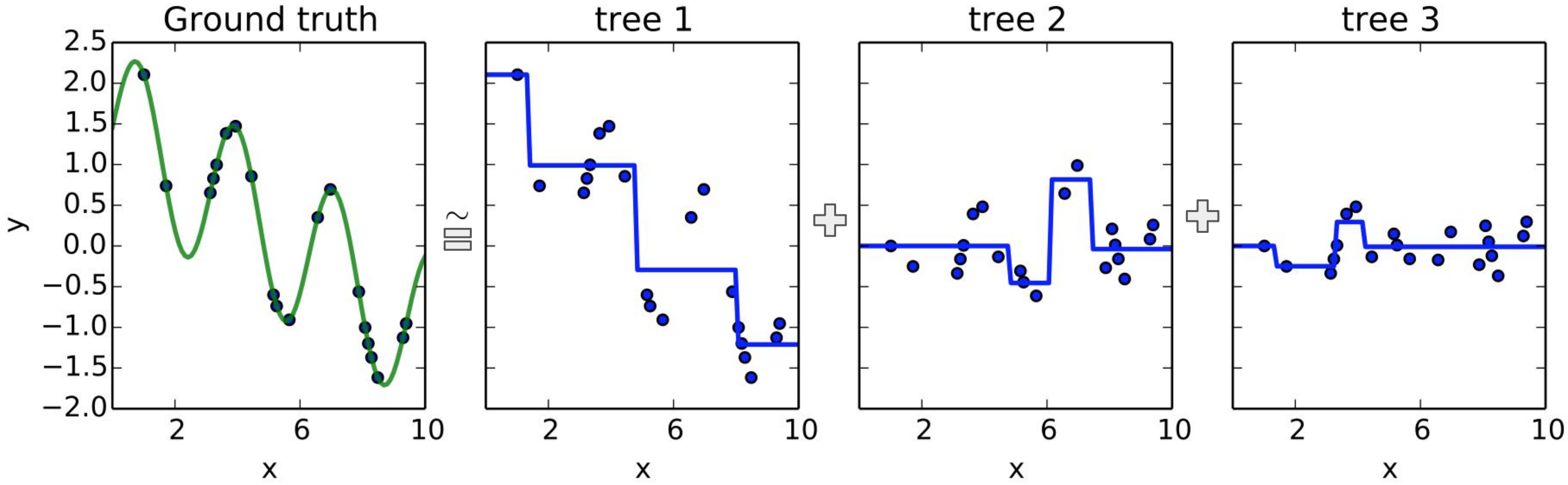
adapted from Wikipedia

Breakout - Part 1

Pair up and ask your partner the following questions:

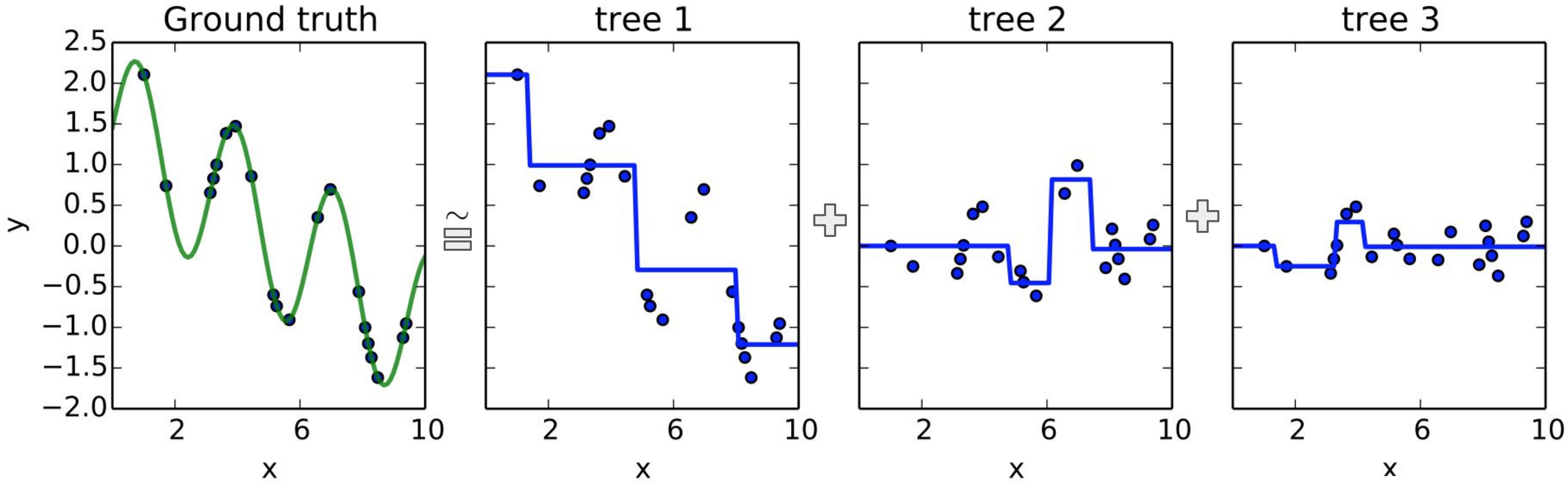
1. What is a *weak learner*?
2. How do bias & variance relate to a *weak* learner?
3. How are *weak* learners used in boosting?
4. Describe how a decision tree can be a *weak* learner.

Breakout - Part 2



1. How many leaf nodes are in *tree 1*? To help visualize, draw the full tree including the values that determine the splits.

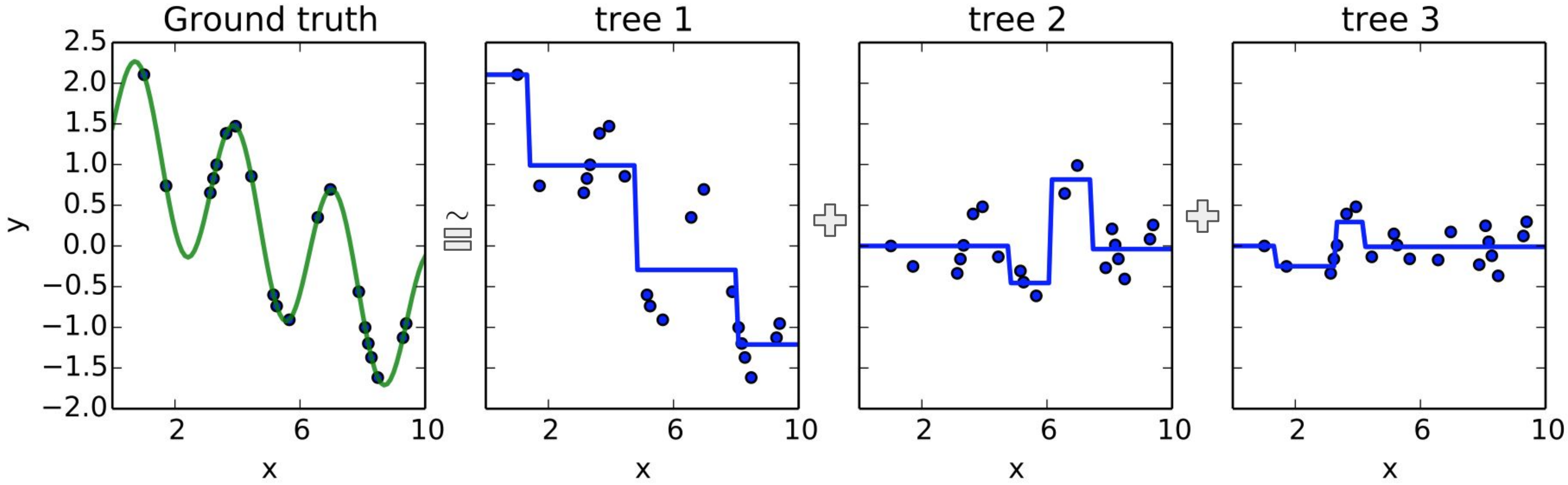
Breakout - Part 2



2. In ISLR, the prediction of a boosted model is represented as $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$

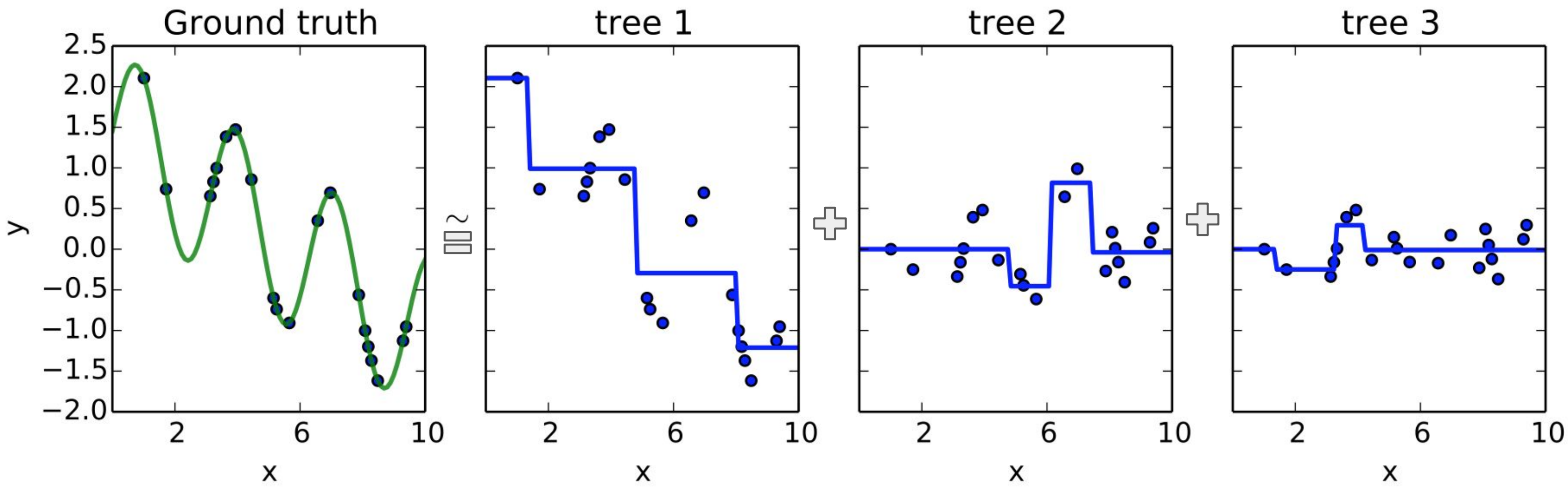
Looking at the diagram above, define the following variables: b B λ $\hat{f}^b(x)$

Breakout - Part 2



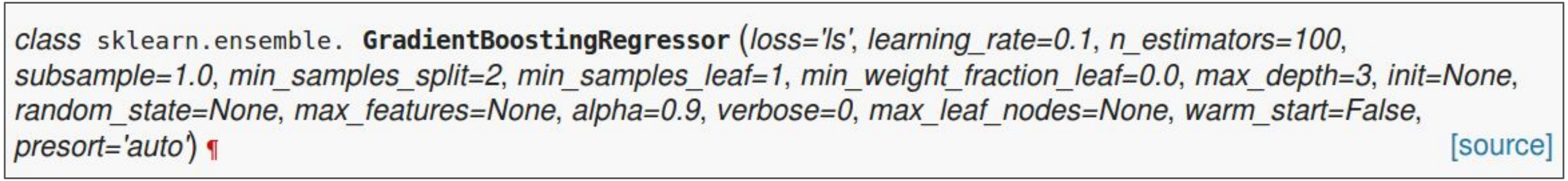
3. Calculate the predicted y-value for new data points x=2 and x=6, given the full 3-tree model above. (You can eyeball this)

Breakout - Part 2

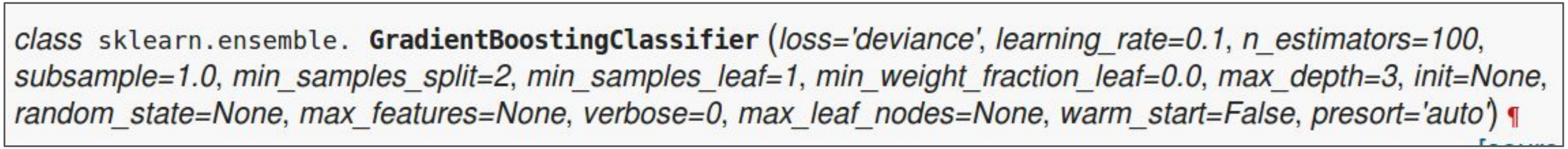


4. There is something that is not technically correct with the depiction above, but served well for this example. What is it?

Gradient boosting in sklearn - and hyperparameters



class sklearn.ensemble. **GradientBoostingRegressor** (*loss='ls', learning_rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, init=None, random_state=None, max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*) ¶ [source]



class sklearn.ensemble. **GradientBoostingClassifier** (*loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*) ¶

Gradient boosted trees have all the same hyperparameters as decision trees, but with a few more:

- `learning_rate`
- `subsample`
- `max_features`

Hyperparameter: `learning_rate`

Also known as the shrinkage parameter, it is applied when one model is added onto another model. Typically ~ 0.2. Highly related to `n_estimators` to get a desired test error. In cross-validation we usually set either the `learning_rate` or `n_estimators` and then vary the other.

No shrinkage:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x),$$

With shrinkage:

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x), \quad 0 < \nu \leq 1$$

where parameter ν is called the "learning rate".

“Empirically it has been found that using small learning rates (such as v < 0.1) yields dramatic improvements in model's generalization ability over gradient boosting without shrinking. However, it comes at the price of increasing computational time both during training and querying: lower learning rate requires more iterations.” - Wikipedia

Hyperparameters: `subsample`

To build randomness similar to random forest methods, we may choose to build each tree on a **`subsample`** of our data.

Unlike in random forests, we sample **without** replacement.

Building in randomness in the above ways creates a method known as Stochastic Gradient Descent, which has been shown to work faster and better in many cases, and provide some regularization.

Recall that Stochastic Gradient Descent, we find the gradient of the cost function of a single randomly selected example at each iteration instead of the sum of the gradient of the cost function of all examples.

Hyperparameters: `max_features and max_depth`

We can might also subset our feature space using **`max_features`** similar to random forests.

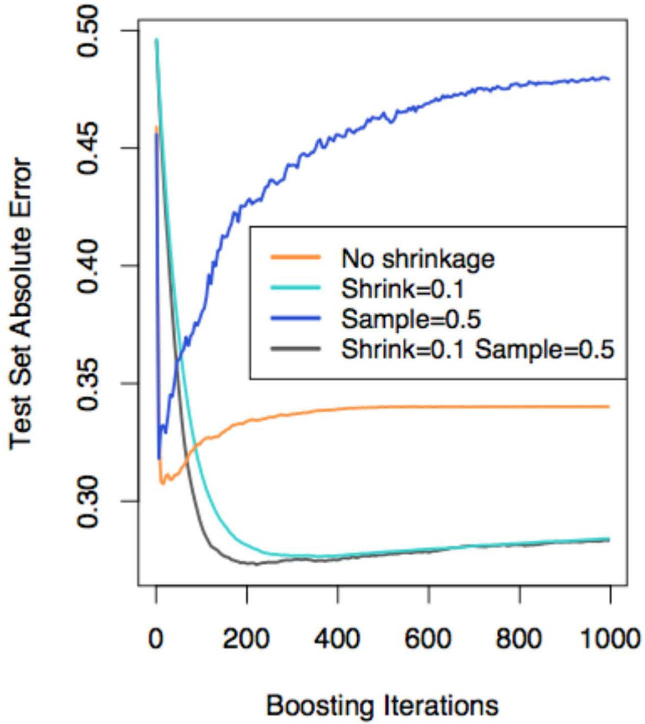
`max_features` also is the number of features to consider when looking for the best split.

`Max_depth` will limit the number of nodes in a tree. This is highly dependent on the interaction of the input variables.

Boosting Tips

Overfitting can be a problem with boosting (debatable). To prevent this:

- Keep the base estimator simple (limit its `max_depth` to 2-8).
- Limit M, the maximum number of iterations. Use staged_predict (a method on a Gradient Boosting Regressor object) to monitor the train and test errors as they evolve with each iteration.
- Use shrinkage
- Use Stochastic Gradient Boosting using `subsample` and `max_features`
- Use large values for `min_samples_leaf` (also limits the depth of the tree)



Boosting - Algorithms of Note

XGBoost* - A top performing algorithm on Kaggle.

*Installation on Mac is very involved - wait until you have time to debug or use Docker

CatBoost - For only categorical features

Jupyter Notebook Demo

`Boosting_drury.ipynb`

Notes:

- Use this for a resource to look at the outputs of the code is already run
- Focus up to ‘Boosting for Classification’
- Also referred to in the PM

Objectives

We are able to...

- Describe conceptually what boosting is
- Describe the Gradient Boosting algorithm
- List Gradient Boosting hyperparameters
- Be able to use GridSearchCV to find the best hyperparameters for a model
- List a couple of *useful* non-sklearn based boosting algorithms