LECTURE 4: CLASSICAL ML: DECISION TREES/ RANDOM FORESTS/BOOSTING

EKATERINA MURAVLEVA

OVERVIEW OF THE COURSE

Lecture 1: General course information, CRISP-DM methodology

Lecture 2: Supervised learning/unsupervised learning. Classification/regression problems. Accuracy metrics (precision, recall, ROC-AUC scores). Concept of loss functions, overfitting / underfitting.

Lecture 3: Classical ML: Linear regression, logistic regression, support vector machine

Lecture 4: Classical ML: Decision trees, random forests, boosting.

Lecture 5: Classical ML: Dimensionality reduction: linear, non-linear methods.

Lecture 6: Classical ML: Clustering methods

Lecture 7: Basic neural networks

Lecture 8: Scalable algorithms

RECAP OF LECTURE 3

- Linear regression
- Logistic regression
- Support vector machine
- Kernel trick

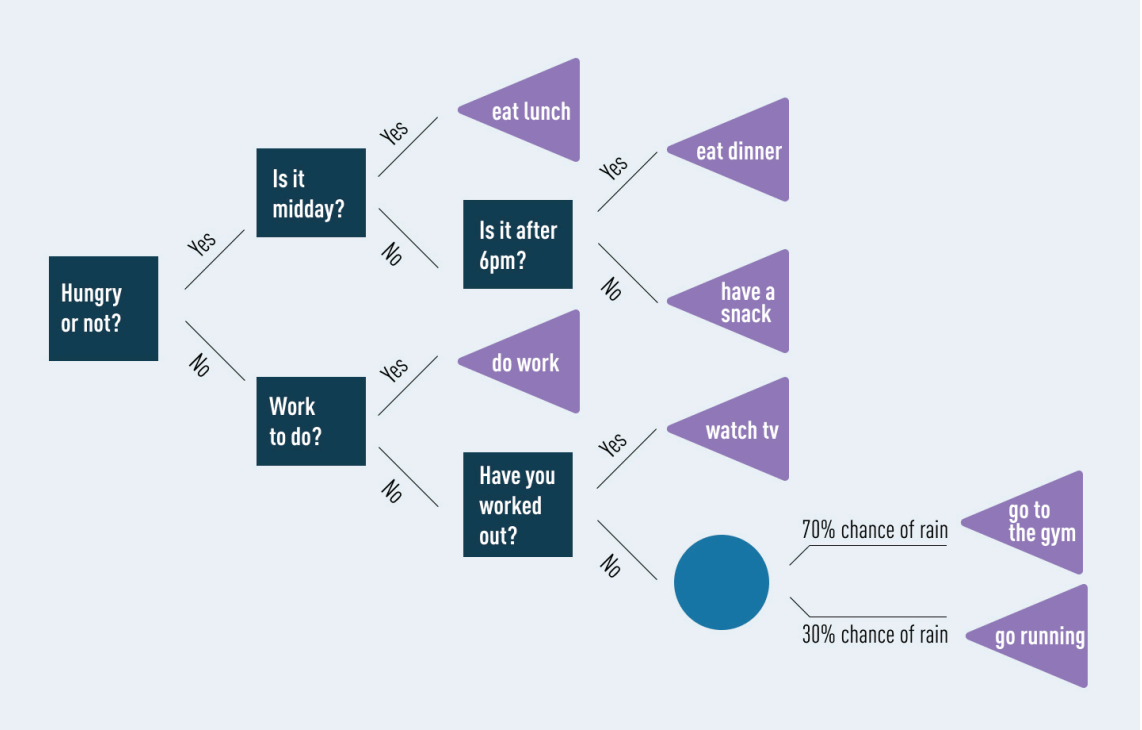
PLAN OF LECTURE 4

- Decision trees
- Random forests
- Gradient boosting

DECISION TREE

The decision tree is a method in machine learning, which tries to mimic the rules for solving different problems

The tree on the right describes the answer to the question ‘what to do’?



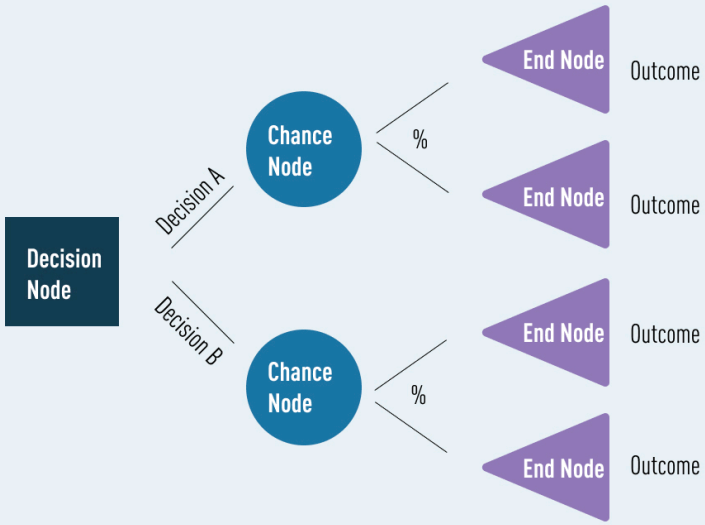
COMPONENTS OF THE DECISION TREE

Decision tree consists of nodes and branches

Root Node: The first node in the path

Leaf Nodes: End of decision path

Internal Nodes: can be decision nodes or chance nodes (where it has branches)



WHAT DECISION TREE CAN BE USED FOR

Decision trees can be used for **classification** and **regression**

The predictions of decision trees are very interpretable and explainable

Question: How to learn a decision tree given some data?

HOW TO TRAIN A DECISION TREE (1)

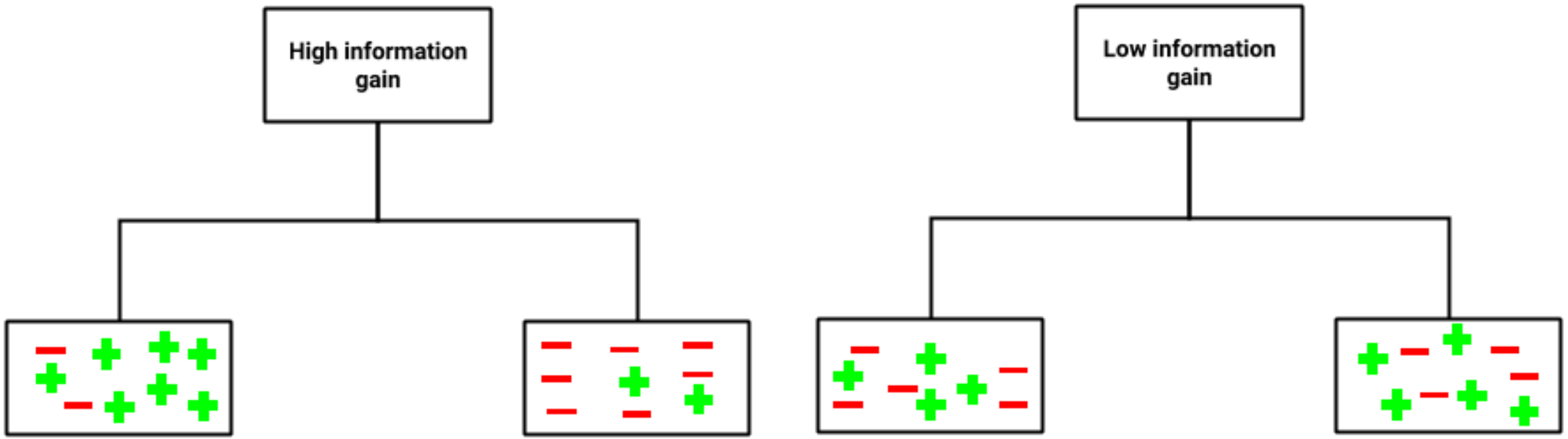
Decision trees are trained from top to bottom (from the root).

In the first step, we only have a root node.

We need to split this root node into child nodes.

How we do that?

LEARNING DECISION TREE



A good split separates the classes. A bad split does not. It is measured using **information gain**

ENTROPY

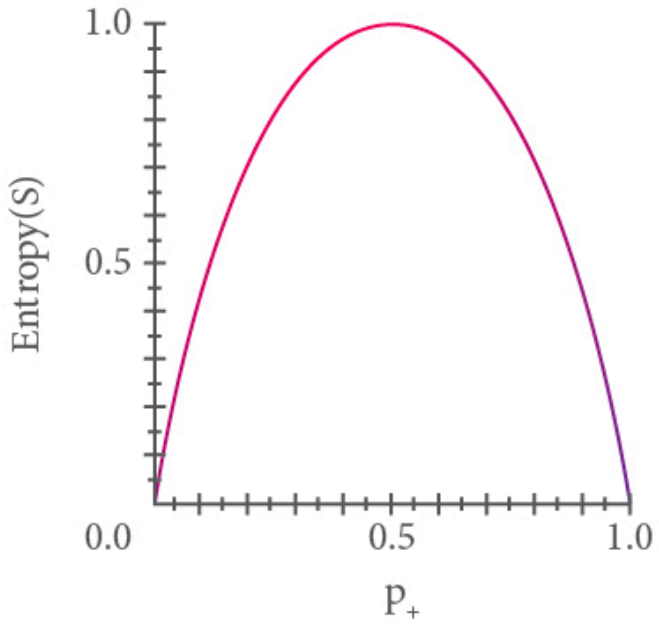
Information gain is defined using **entropy**

Entropy is defined as $\sum_i -p_i \log p_i$, where p_i is probability of class i

For binary classification, we have two classes, thus

$$\text{Entropy} = -p_+ \log p_+ - p_- \log p_-$$

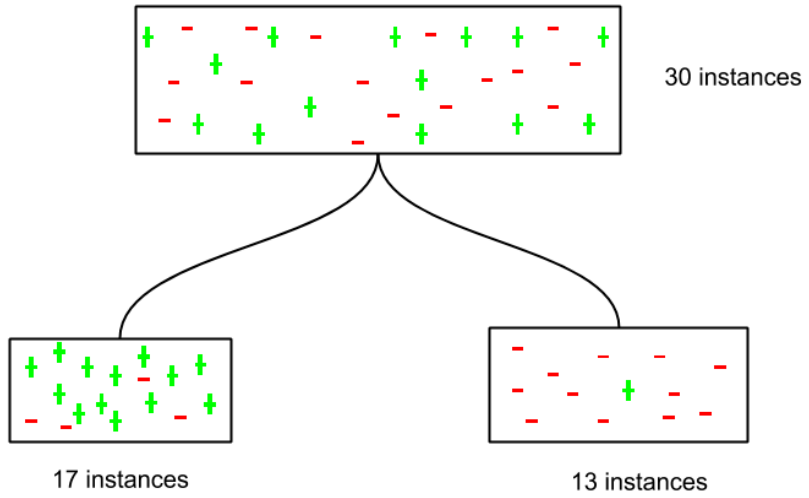
Question: If all samples in a node belong to the same class, what is the entropy?



INFORMATION GAIN

Given the split of the Parent node into Children, we can define the Information Gain as

Information Gain = Entropy(Parent Node) - AverageEntropy(Children)



LEARNING DECISION TREE

Training data consists of many instances of the following kind:

Instance 1	attribute1	attribute2	...	attributeK	class
Instance 2	attribute1	attribute2	...	attributeK	class

i.e., objects with attributes.

We assign all instances to the root node.

Then **for each attribute:**

Partition all the data instances (samples) at the node **by the value of a single attribute**

Compute how **good is the split** using Information gain (or other metrics)

After the split: if the child has only one class, stop.

If not, **recursively split.**

PROBLEMS WITH INFORMATION GAIN

The problem with information gain is that it may create **unbalanced trees:**

some children have small number of samples.

Another approach used in CART (Classification and Regression Trees)

Is the following ‘goodness function’: t is the root node, s is the split, t_L, t_R are left and right children, simultaneously and P_L, P_R are sizes of the split.

$$\varphi(s \mid t) = 2P_L P_R \sum_{j=1}^{\text{class count}} \left| P\left(j \mid t_L\right) - P\left(j \mid t_R\right) \right|$$

WORKING WITH CONTINUOUS VARIABLES (REGRESSION TASK)

The criteria described before are for **discrete target variables (classification task)**

For regression task, we need to **select another criteria**.

The standard approach is to use **variance** of the estimated variable. If the variance is small, the prediction is good. It is defined as

$$I_V(N) = \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2} \left(y_i - y_j \right)^2 - \left(\frac{|S_t|^2}{|S|^2} \frac{1}{|S_t|^2} \sum_{i \in S_t} \sum_{j \in S_t} \frac{1}{2} \left(y_i - y_j \right)^2 + \frac{|S_f|^2}{|S|^2} \frac{1}{|S_f|^2} \sum_{i \in S_f} \sum_{j \in S_f} \frac{1}{2} \left(y_i - y_j \right)^2 \right)$$

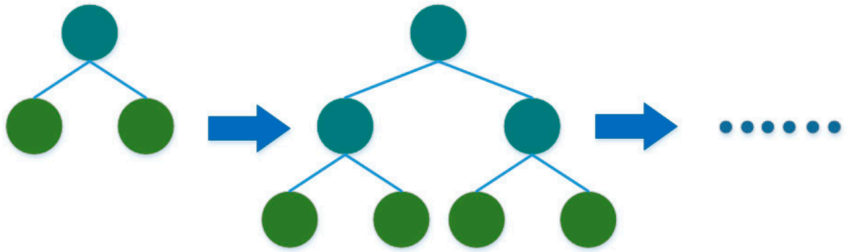
Where S, S_t, S_f are the set of presplit sample indices, set of sample indices for which the **split test** is true, and set of sample indices for which the split test is false, respectively.

DIFFERENT WAYS OF GROWING TREES

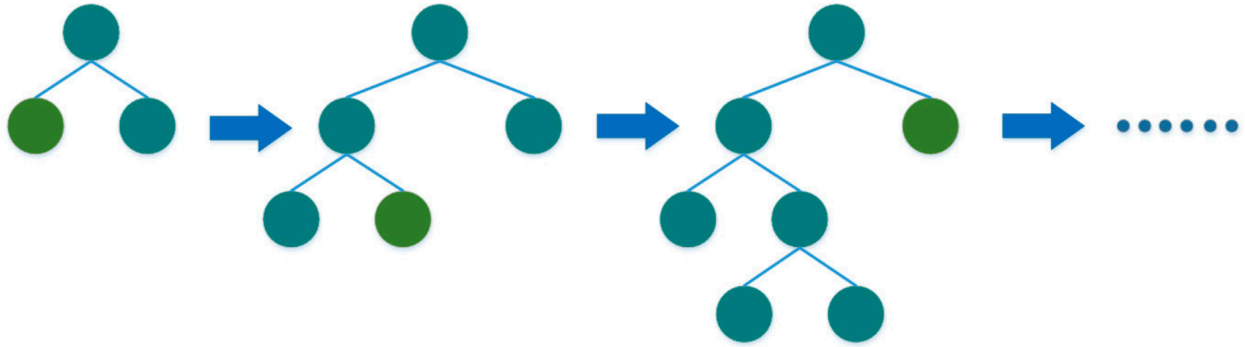
For practical implementation, one can choose different order of growing trees:

Level-wise (or depth-wise) and leaf-wise

They are shown on the pictures



growth of level-wise tree



growth of leaf-wise tree

DECISION TREES: SUMMARY

Decision trees are easy to interpret.

Decision trees tend to overfit the training data, especially if there size of training data is small.

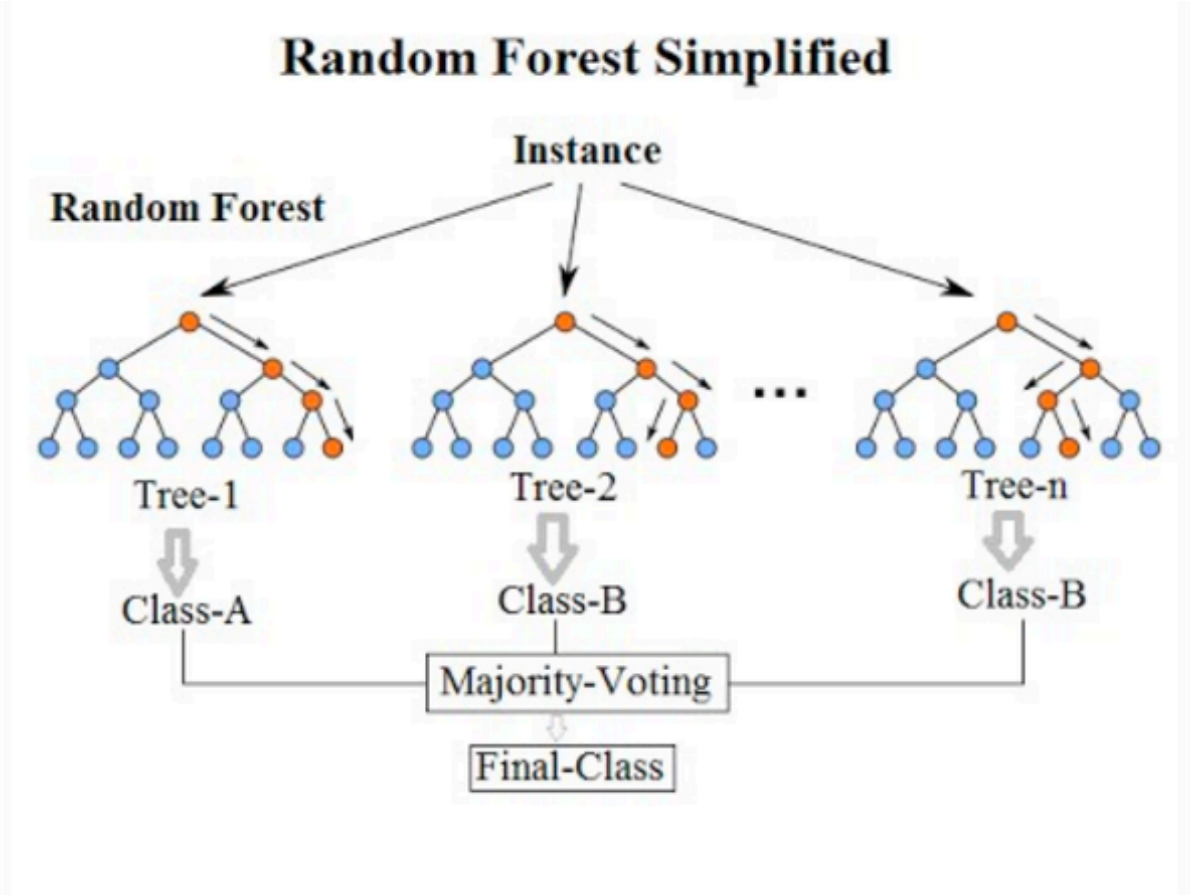
A solution to this is creating not a single decision tree, but **ensemble of them.**

RANDOM FOREST

Random forest is the set of decision trees

For **regression** their answers are averaged

For classification their answers are selected using **majority voting**



HOW TO BUILD RANDOM FOREST

Each decision tree is trained **independently** on:

1. Random subsets of the datasets (maybe intersecting)
2. Each split in each tree is selected from **randomly selected features**
3. The tree is built until each node has only samples from one class.

The main principle of ensembling: base models should be good and diverse.

PROPERTIES OF RANDOM FOREST

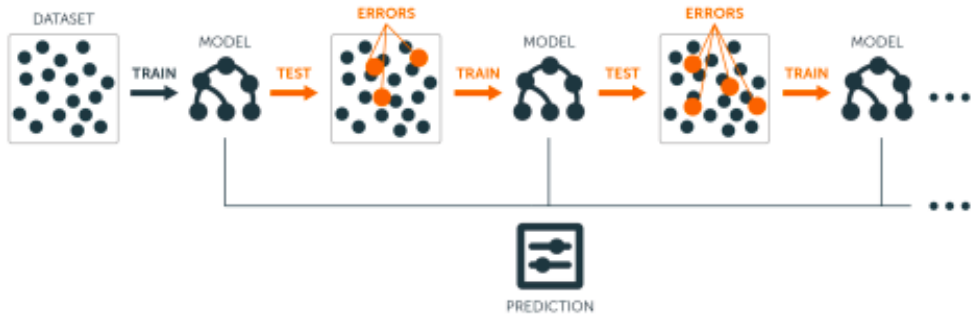
1. (+) Generalize better than decision trees
2. (-) Are not interpretable as easy as decision trees.
3. (+) Actively used as baselines

BOOSTING

Super-powerful idea in classical machine learning is **boosting**

We train a simple model (i.e., decision tree) and then train another model to correct the errors.

It is a part of a general robustness theory for AI models: they always will make mistakes.



GRADIENT BOOSTING (1)

Let (x_i, y_i) be our training set.

Let $a(x)$ be the algorithm (model) that predicts $a(x_i) \approx y_i$ using certain loss function $L(y, a)$

Let $b(x)$ be the **corrector model** that tries to solve

$$a(x_i) + b(x_i) = y_i$$

What dataset we should train b on?

GRADIENT BOOSTING (2)

Lets write down the optimization problem as

$$\sum_{i=1}^{N} L(y_i, a(x_i) + b_i) \to \min$$

The function decays as fast as possible in the direction of the anti-gradient.

Thus it is **quite natural to train** b **on the dataset** $(x_i, -L'(y_i, a(x_i))$,

In other words, $b(x_i) \approx b_i = -L'(y_i, a(x_i))$

GRADIENT BOOSTING (3)

Suppose, $b(x_i) \approx -L'(y_i, a(x_i))$

Then the total prediction will be $a(x_i) + b(x_i) = a(x_i) - L'(y_i, a(x_i))$

Depending on the loss function, we can add a learning rate parameter here, yielding

$a(x_i) + \eta b(x_i) \approx a(x_i) - \eta L'(y_i, a(x_i))$

For sufficiently small, η we will have $L(y_i, a(x_i) + \eta b(x_i))) < L(y_i, a(x_i))$

GRADIENT BOOSTING (4)

We are training $b(x_i) \approx -L'(y_i, a(x_i))$

Example 1: $L(y, a) = \frac{1}{2}(y - a)^2$, $-L'(y, a) = y - a$, thus for quadratic function we are approximating the error of approximation

Example 2: Logistic loss $L(y, a) = \log\left(1 + e^{-y \cdot a}\right), a \in (-\infty, +\infty), y \in \{-1, +1\}$

The anti-gradient differs a lot from approximation of the error.

GRADIENT BOOSTING (5)

Suppose we have learned the correct model, $b_1(x_i) \approx -L'(y_i, a(x_i))$

We can do this multiple times!

$$a_n(x) = \sum_{t=1}^{n} b_t(x)$$

Suppose we have algorithm $a_t(x)$, and we train $b_t(x)$ on the set $(x_i, -L'(y_i, a_t(x_i)))$

GRADIENT BOOSTING (6)

Instead of the gradient direction $b_1(x_i) \approx -L'(y_i, a(x_i))$

We can try to fit the Newton direction:

$$b_1(x_i) \approx -\frac{L'(y_i, a(x_i))}{L''(y_i, a(x_i))}$$

This approach is used in the famous XGBoost library.

STOCHASTIC VERSION

We can use the same idea as in the random forests and

train correction models on random subsets of the initial dataset.

- Can improve the quality of the ensemble
- Reduce the time of computations

SOTA BOOSTING LIBRARIES

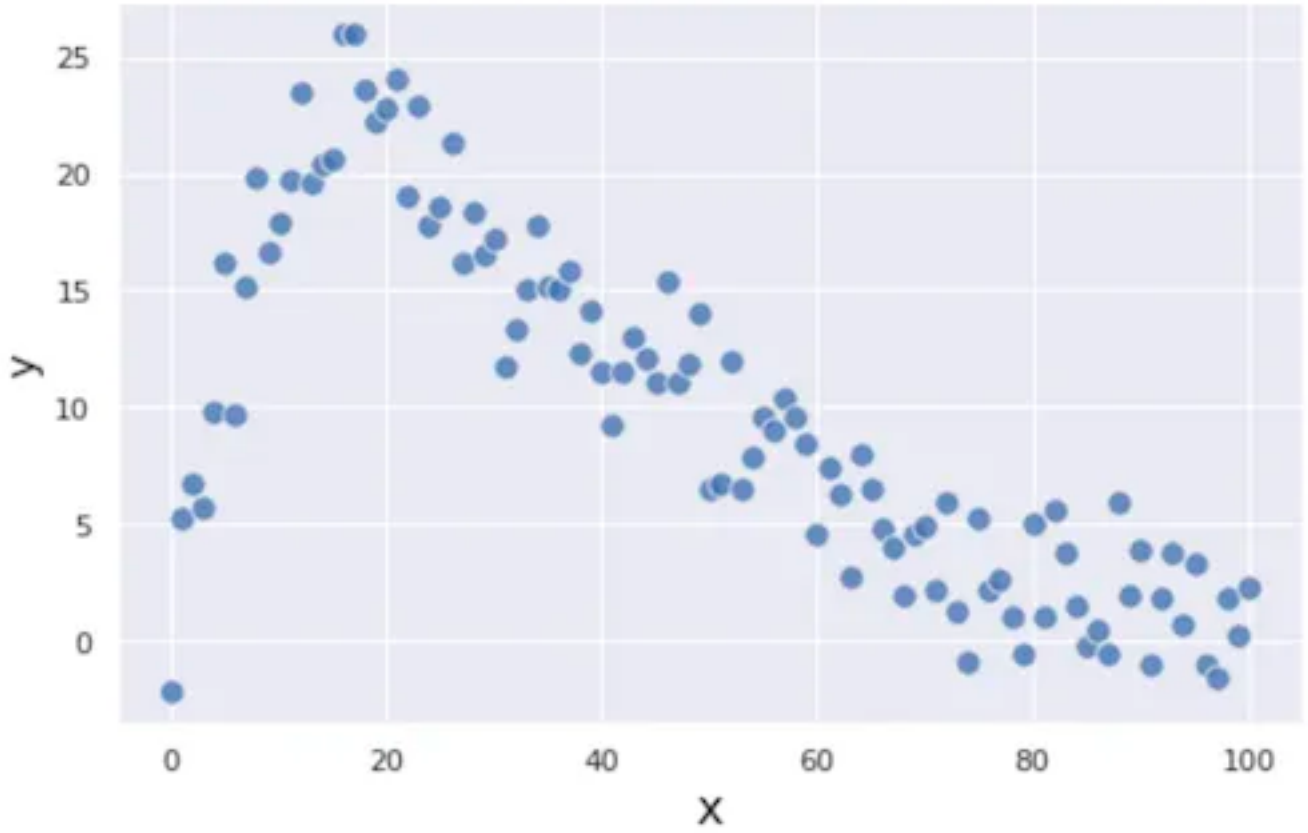
- Boosting is used with decision trees
- Modern libraries are XGBoost, CatBoost, LightGBM
- They differs in implementation, used algorithms for ‘growing trees’, types of boosting, etc

GRADIENT BOOSTING: EXAMPLE

Consider gradient boosting for the following regression problem.

We need to predict y from x,

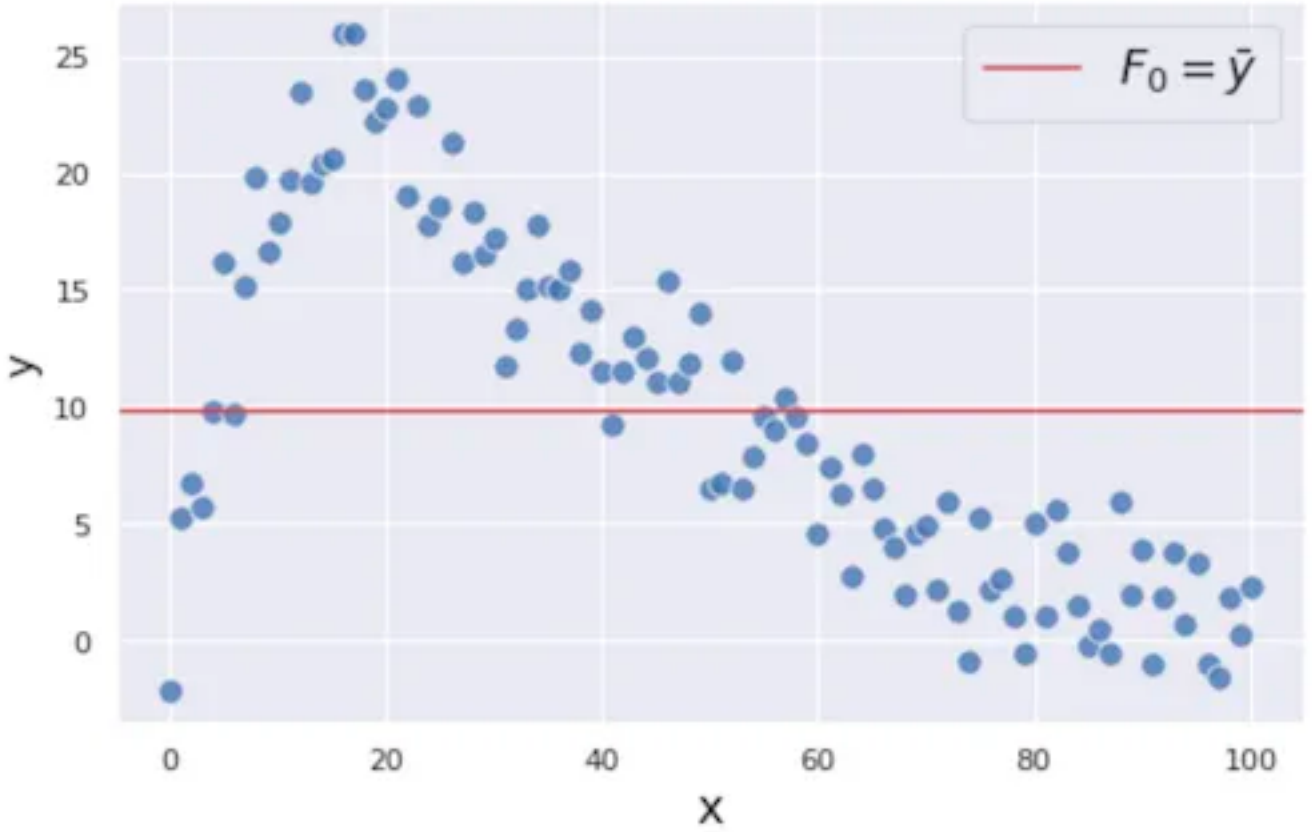
$y = a(x_i)$



FIRST STEP

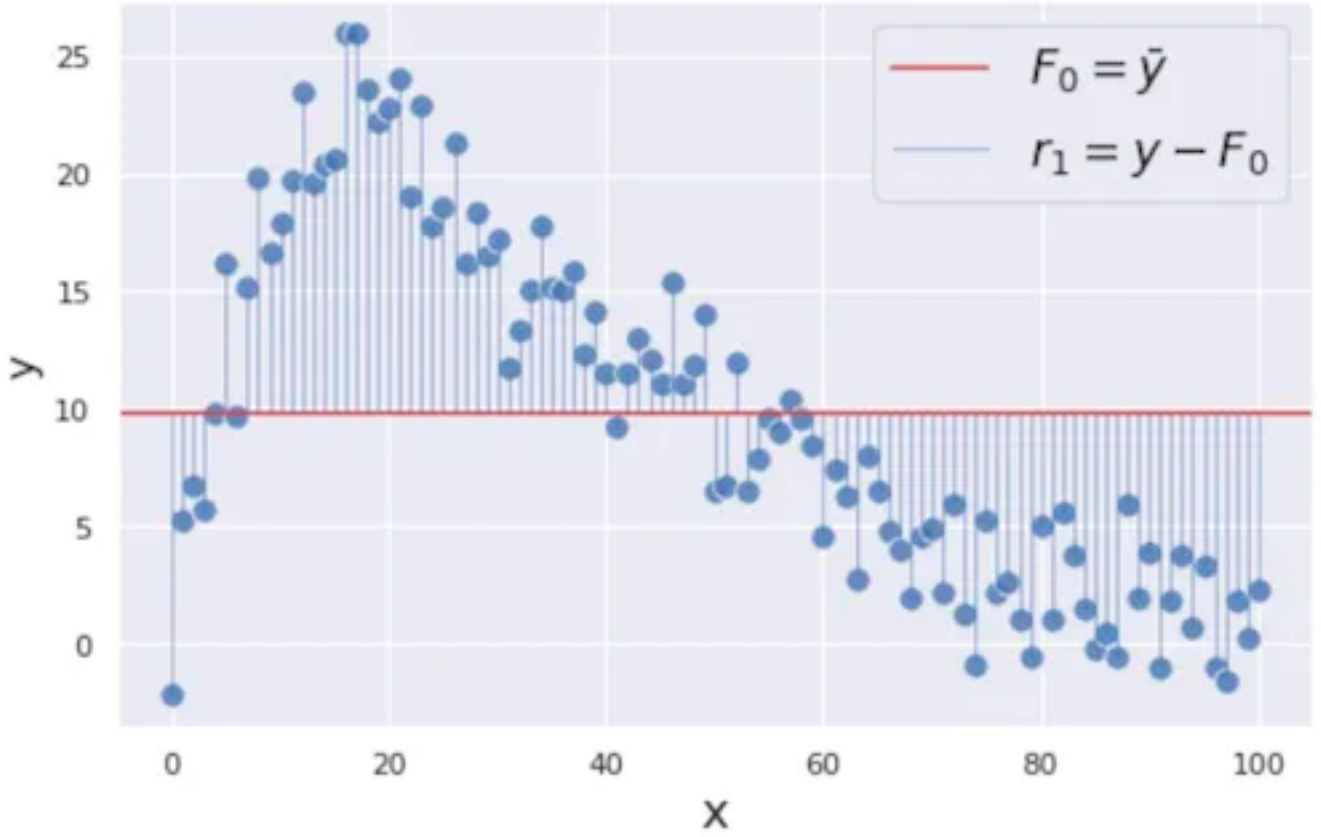
The first step is to create a simple model (naive prediction).

As an initial prediction, we will use the **mean value**



COMPUTING RESIDUALS

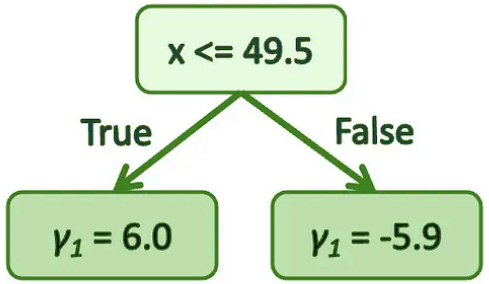
The naive prediction is very inaccurate, so we need to approximate the residual (remembe the anti-gradient formulation for the square loss function).

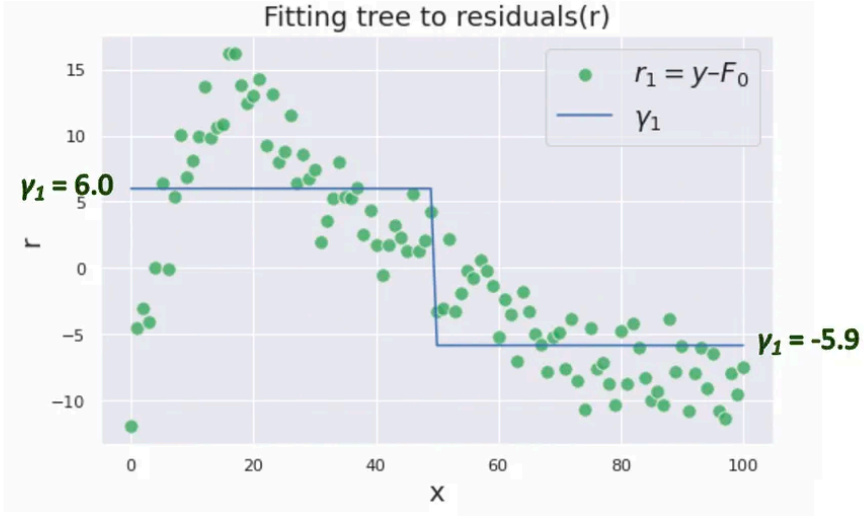


COMPUTING RESIDUALS

We will now build very simple **decision trees** as corrector models

They have one split and two leaf nodes.





Typically, gradient boosting uses trees from 8 to 32 leaf nodes.

We will use the tree that predicts two values 6.0, -5.9

This prediction is added to the initial prediction:

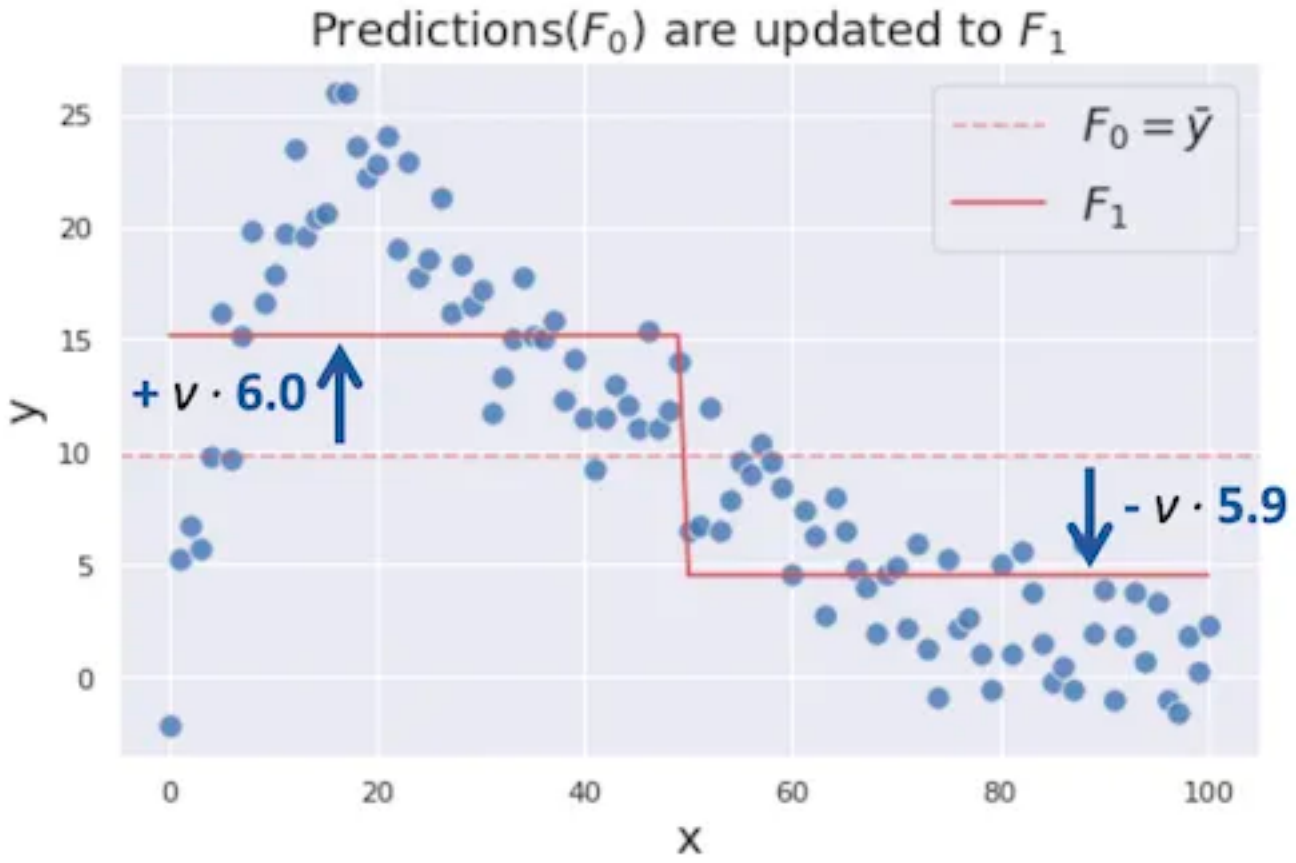
$$F_1 = F_0 + \nu\gamma_1$$

In this example, we can take $\nu = 0.9$

NEW APPROXIMATION

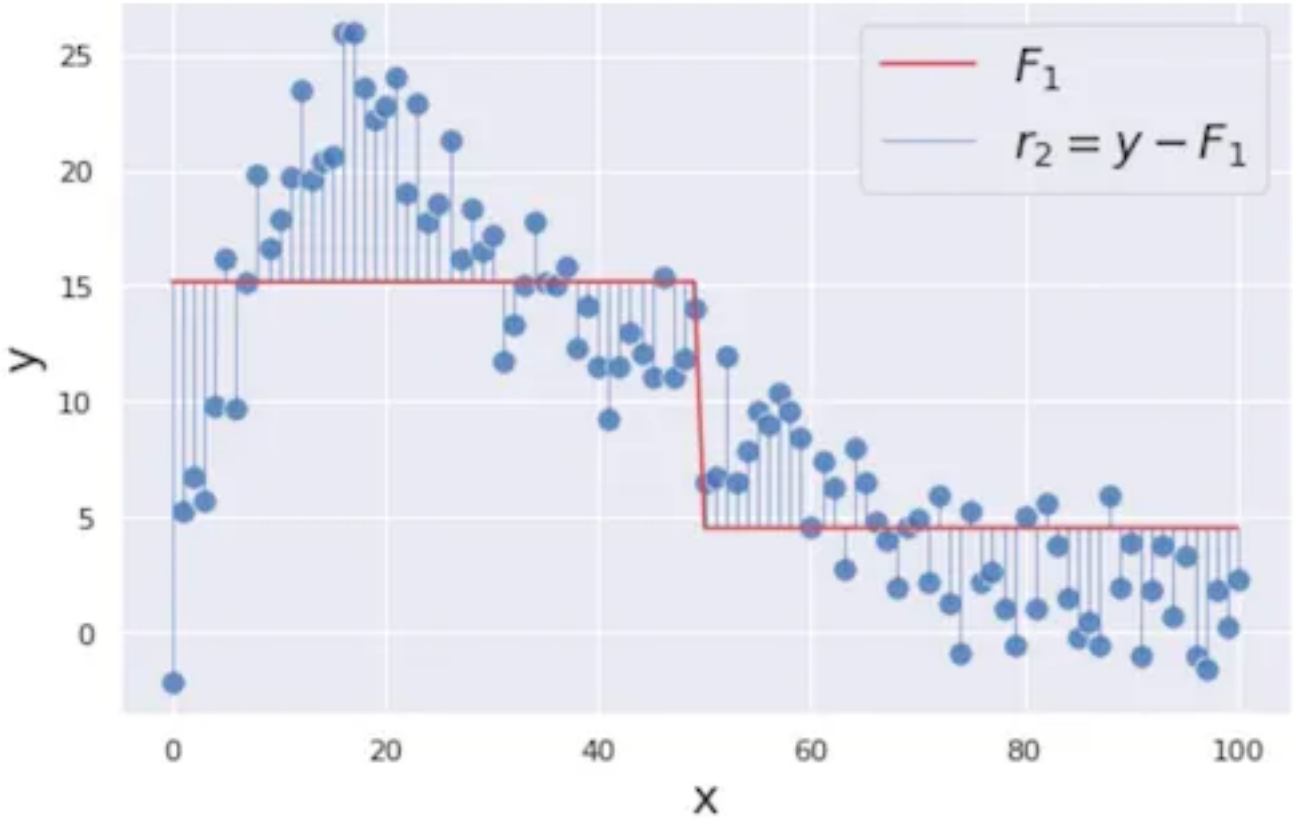
After update, our combined predictio
becomes

$$F_1 = \begin{cases} F_0 + \nu 6.0, & x \leq 49.5, \\ F_0 - \nu 5.9 & \text{otherwise} \end{cases}$$



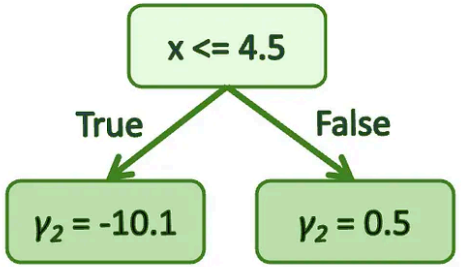
NEW RESIDUALS

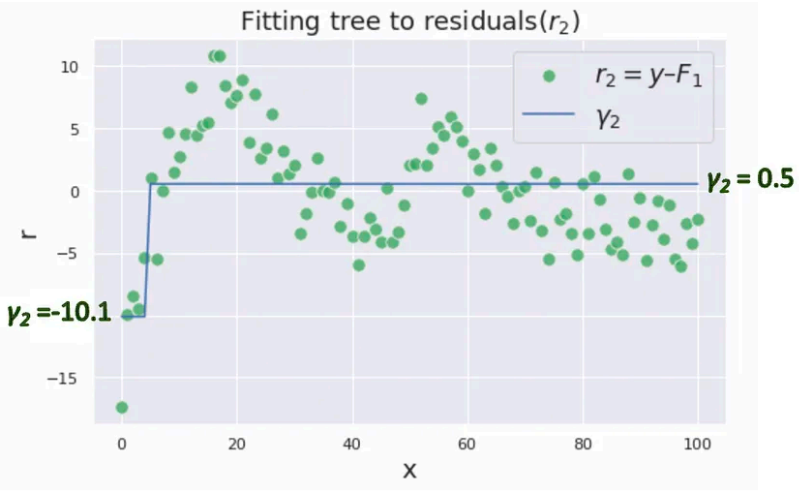
The updated residuals look like this



NEXT STEP

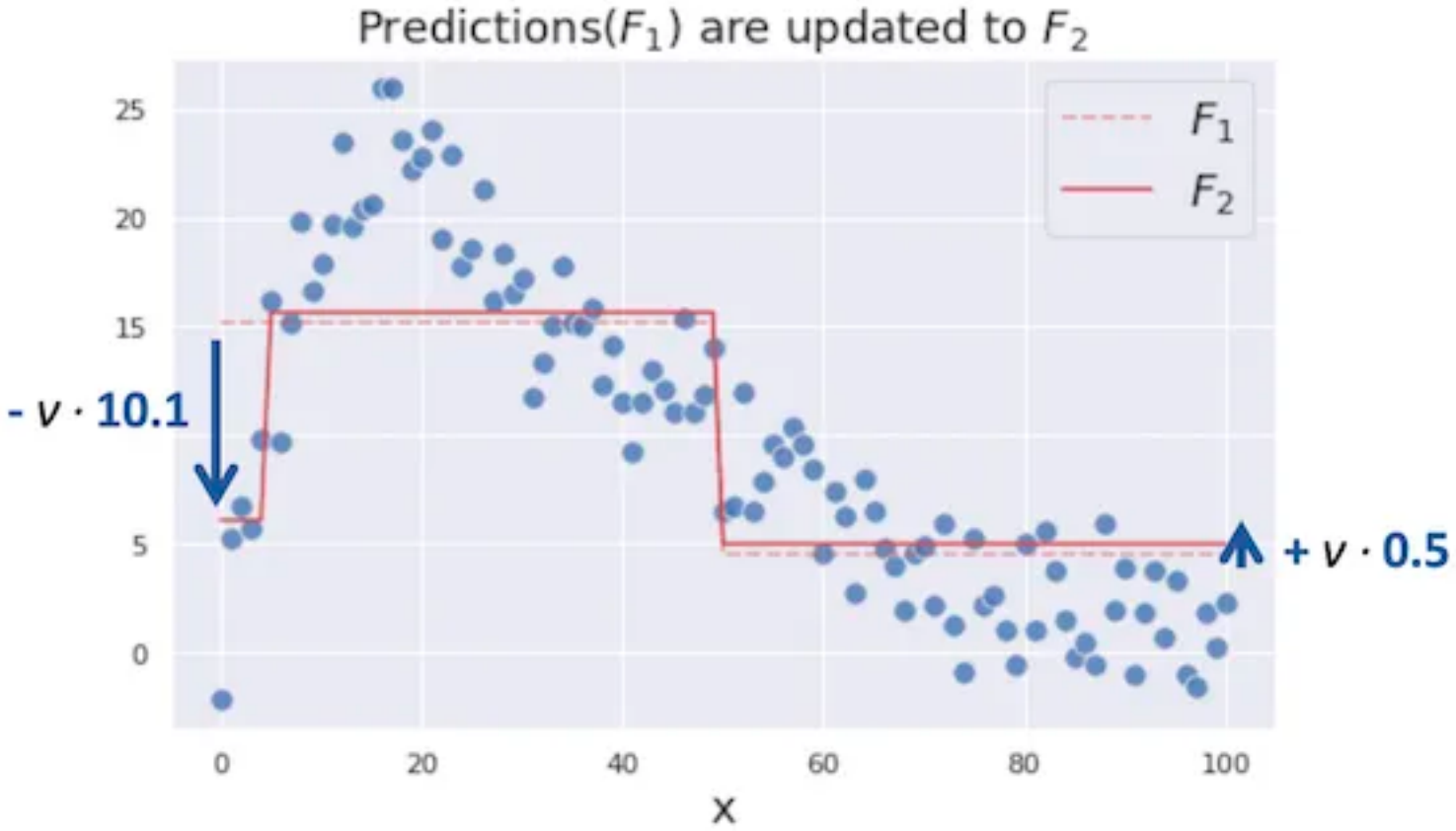
In the next step, we create the regression tree again using x as a feature





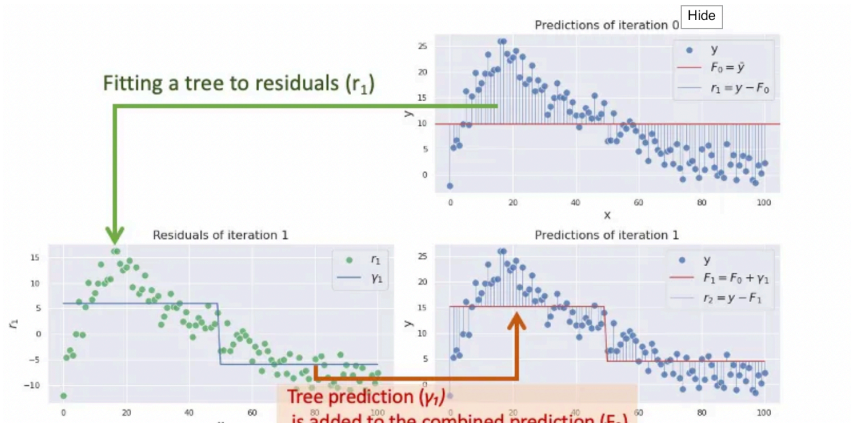
NEXT APPROXIMATION

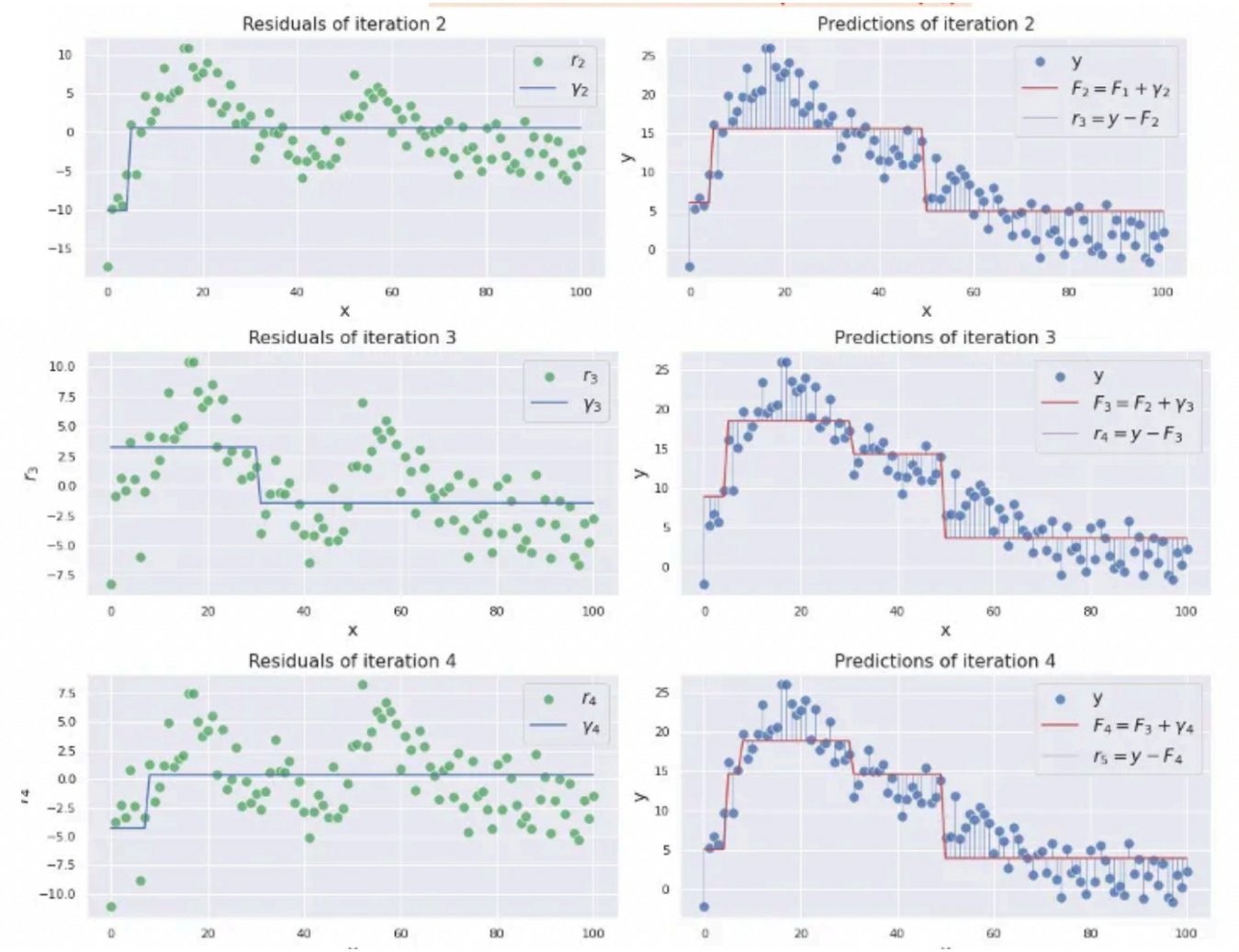
The next approximation has the following form



AFTER ITERATIONS…

The next approximation has the following form





RECAP OF LECTURE 4

- Decision trees
- Random forests
- Gradient boosting

NEXT LECTURE

Classical ML:

Dimensionality reduction: linear and non-linear methods