

# Ensembles

Following our Recipe for Machine Learning, we may try out several models before deciding on the final one.

Is a single "best" model really best ? Is there an alternative ?

By combining models with independent errors, we may be able to construct a combined model whose accuracy is better than the best individual model.

The combined models are called an *Ensemble*.

The individual models

- May be of different types:
  - Decision Tree, Logistic Regression, KNN
- May be of the *same* type, with different parameters/hyper-parameters:
  - Decision Trees of different depths or different features
  - Regression with polynomial features of different degrees

When the individual models are of the same type

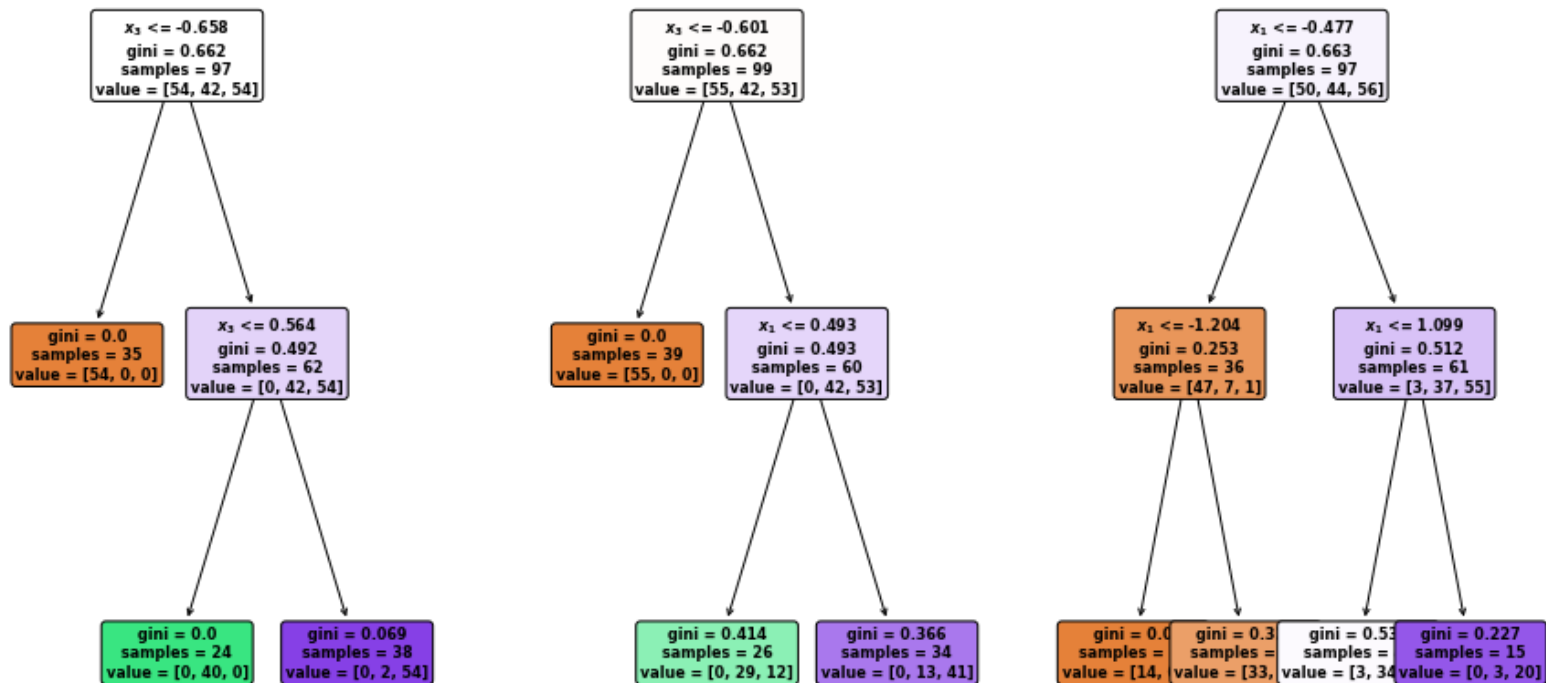
- Each individual models is trained on a *different* subset of the training examples
- This enables the individual models to produce different results
- Makes them more robust to outliers

We will shortly explain how the subsets are chosen.

Here is an Ensemble of individual models of the same type: Decision Trees

In [7]: fig\_ens

Out[7]:



The individual models are usually quite simple and restricted.

- They are *weak learners*: accuracy only marginally better than chance
- But combine to create a *strong learner*.

If the prediction of an ensemble of  $M$  binary classifiers is based on a "majority vote"

- The prediction is incorrect only if  $m' \geq \lceil M/2 \rceil$  classifiers are incorrect
- The probability of a particular set of  $m'$  models of equal accuracy  $A$  all being incorrect is  $(1 - A)^{m'}$
- There are

$$\binom{M}{m'}$$

combinations of  $m'$  models

- So the probability of a correct ensemble prediction when  $m'$  classifiers are incorrect is

$$1 - \binom{M}{m'} * (1 - A)^{m'}$$

which tends to 1 as  $M$  increases.

The power of Ensembles comes via the size of  $M$ .

Ensembling is independent of the types of the individual models

- A meta-model that can combine many different types of individual models
- Under the assumption of **independent** errors
- Often applied in competitions



# Ensemble prediction

Each individual model comes up with a prediction for the target  $\hat{\mathbf{y}}^{(i)}$  of example  $i$ , given features  $\mathbf{x}^{(i)}$ .

Let  $p_{(t),c}^{(i)}$

- Denote the probability predicted by the  $t^{th}$  individual classifier
- That target  $\mathbf{y}^{(i)}$  is in category  $c \in C$
- Given features  $\mathbf{x}^{(i)}$

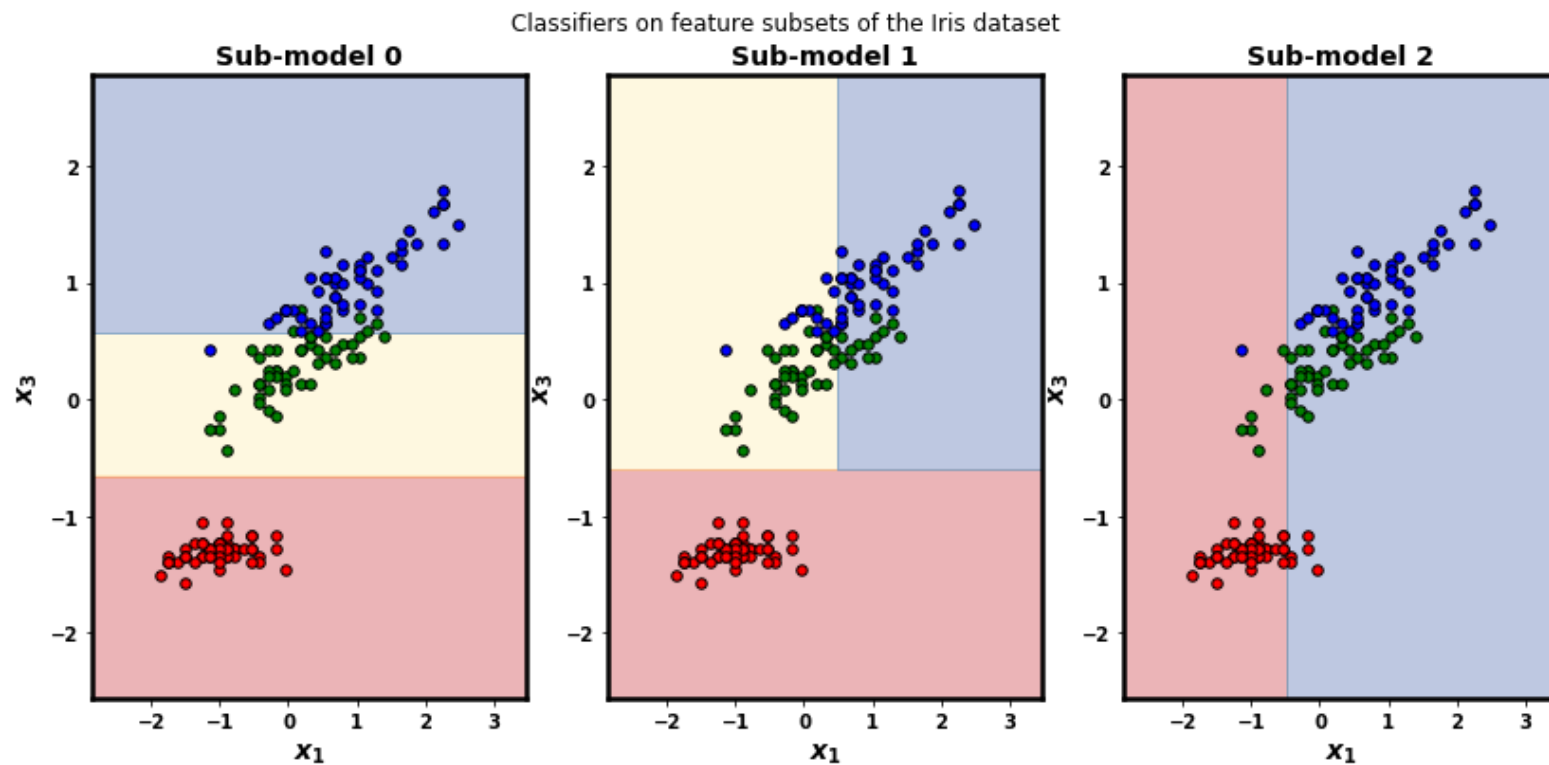
The class predicted by the ensemble is the one with highest average (across individual models) probability

$$\hat{\mathbf{y}}^{(i)} = \operatorname{argmax}_c \sum_{t=1}^M p_{(t),c}^{(i)}$$

Returning to the Ensemble of Decision Trees example, we can plot the decision boundary of each individual model

In [8]: `fig_submodels`

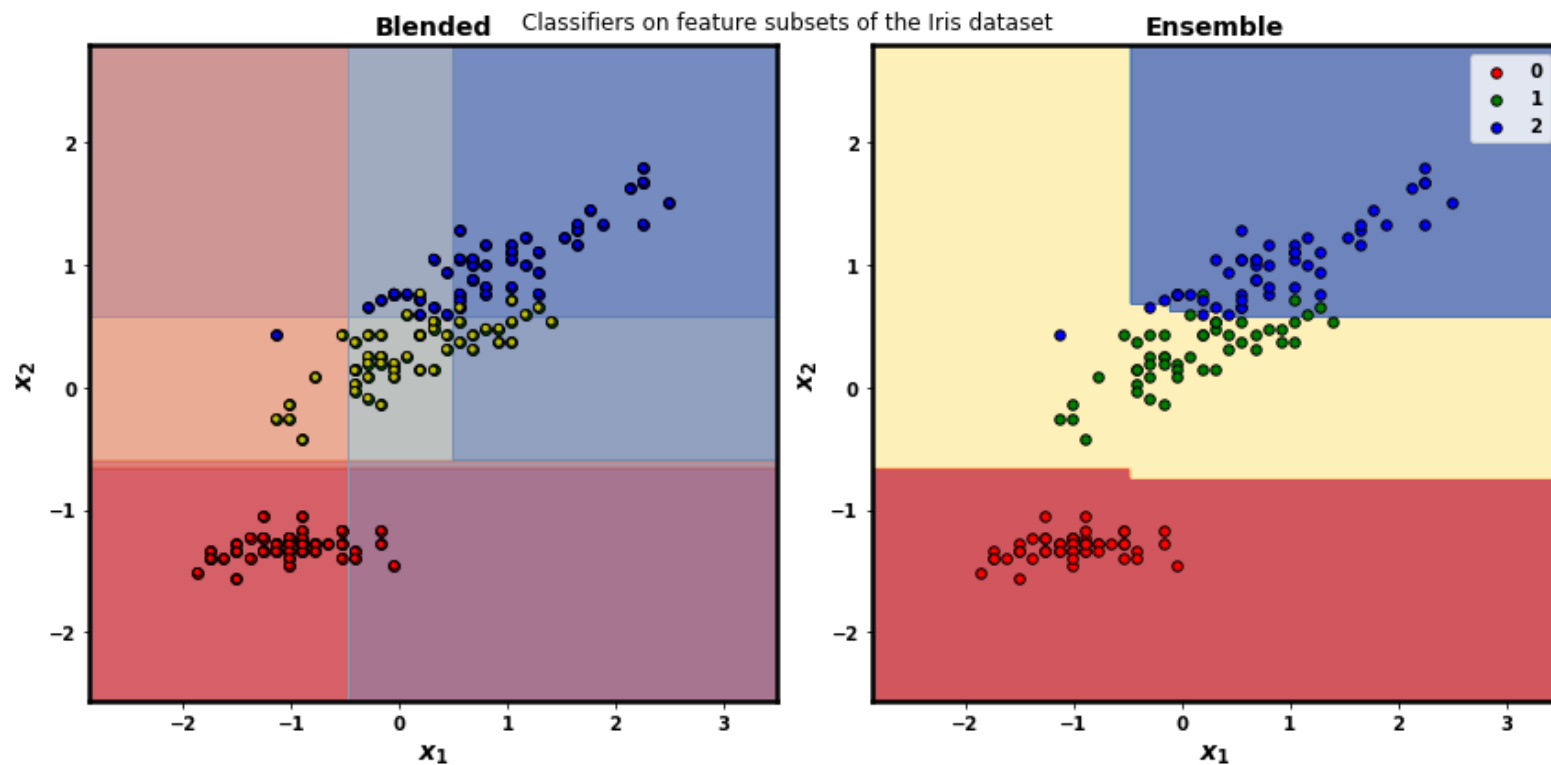
Out[8]:



By superimposing these boundaries on top of one another, we can visualize the "vote"

In [9]: fig\_sum

Out[9]:



- The left plot is the super-position
- The right plot is the final boundary of the ensemble

You can see that the combination of the weak learners does a pretty good job !

# Bagging, Bootstrapping

One way to construct multiple weak learners of the *same* type of model

- Is to train each individual model on a *restricted* set of training examples

Because each individual model is trained on different examples, the predictions made by each are hopefully somewhat independent.

Given the full set of training examples

$$\langle \mathbf{X}, \mathbf{y} \rangle = [\mathbf{x}^{(i)}, \mathbf{y}^{(i)} | 1 \leq i \leq m]$$

we construct a restricted set of examples

$$\langle \mathbf{X}_{(t)}, \mathbf{y}_{(t)} \rangle$$

on which to train the  $t^{th}$  individual model



The restricted set is constructed by

- Selecting  $m$  examples at random from  $\langle \mathbf{X}, \mathbf{y} \rangle$
- *With replacement*
- So it is possible for an example  $i'$  to appear more than once in  $\langle \mathbf{X}_{(t)}, \mathbf{y}_{(t)} \rangle$

This process is called *bootstrapping* and results in

- $\langle \mathbf{X}_{(t)}, \mathbf{y}_{(t)} \rangle$   
 $= [\mathbf{x}^{(i')}, \mathbf{y}^{(i')} | i' \in \{i_1, \dots, i_m\}]$
- Where  $i'_1, \dots, i'_m$  are the indices of the  $m$  chosen examples

If each of the  $m$  examples in  $\langle \mathbf{X}, \mathbf{y} \rangle$  is chosen with equal probability  $\frac{1}{m}$

- The probability of a particular example  $i$  **not** being in  $\langle \mathbf{X}_{(t)}, \mathbf{y}_{(t)} \rangle$  is

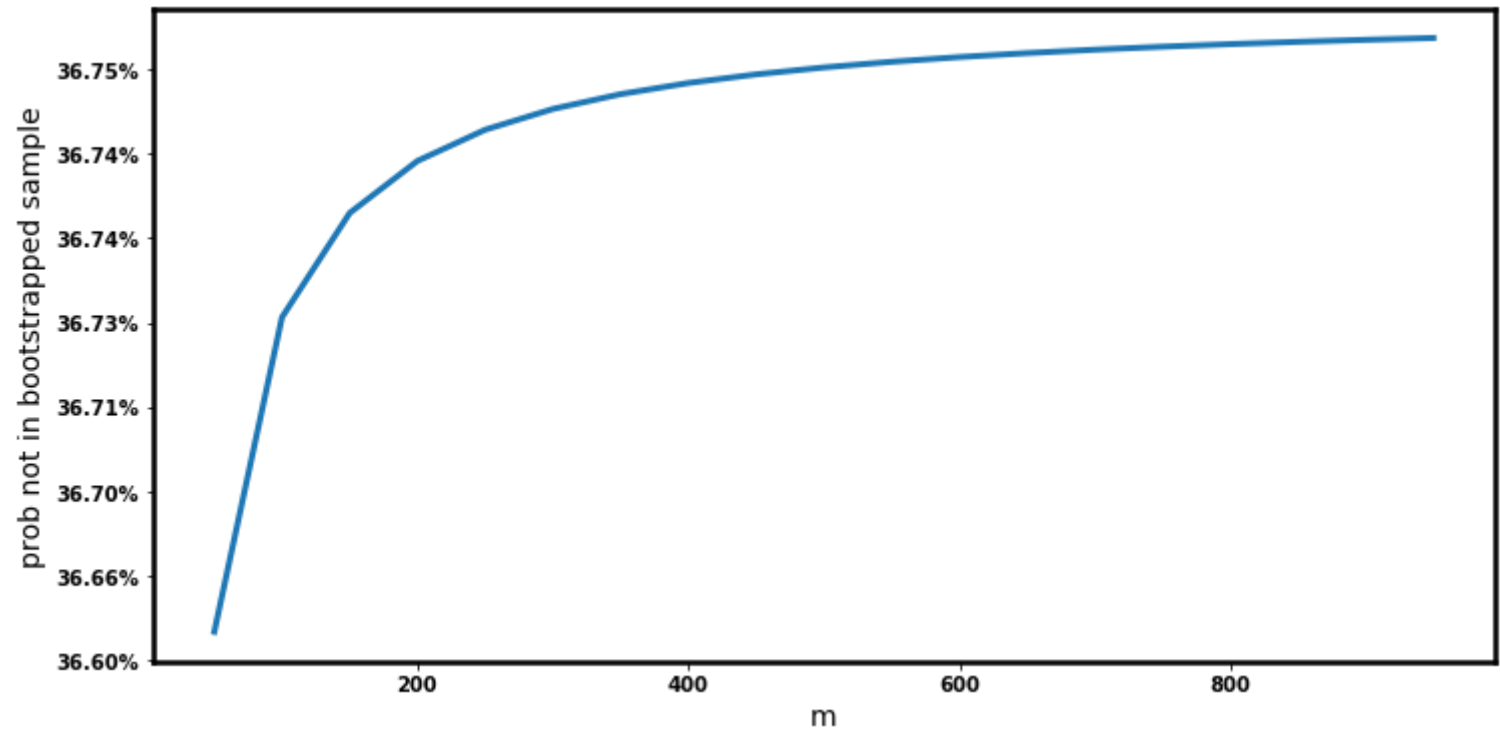
$$\left(1 - \frac{1}{m}\right)^m$$

```
In [10]: m = np.arange(50, 1000, 50)
p = (1 - 1/m)**m

fig, ax = plt.subplots(1,1, figsize=(12,6))
_ = ax.plot(m, p)
_ = ax.set_xlabel("m")
_ = ax.set_ylabel("prob not in bootstrapped sample")
_ = ax.set_yticklabels( [ "{:.2%}".format(y) for y in p])
plt.close(fig)
```

In [11]: fig

Out[11]:



Thus about 63% of the examples in the bootstrapped set are duplicates.

The weak learner can't overfit to any example that is not in its training set.

The process of

- Bootstrapping restricted training examples
- Training individual models on the bootstrapped examples
- Aggregating model predictions into a single prediction

is called *bagging* and each individual training set is called a bag

Bagging has a nice side-effect

- About 37% of the full set of examples are not present in a given bag
- Called *out of bag*

The out of bag examples thus can be used to test out of sample prediction !

# Random Forests

A Random Forest

- Is a collection of Decision Trees
- Of restricted power (weak learners)
- Created by Bagging



The learners are made weak by

- Training on a bootstrapped subset
- By limiting the depth of the Decision Tree
- By limiting the choice of feature on which to split a node
  - To a random subset of all features

The result is that the individual models (Decision Trees) are relatively independent.

# Boosting

There is another approach to creating ensembles of weak learners.

The method is called *boosting*

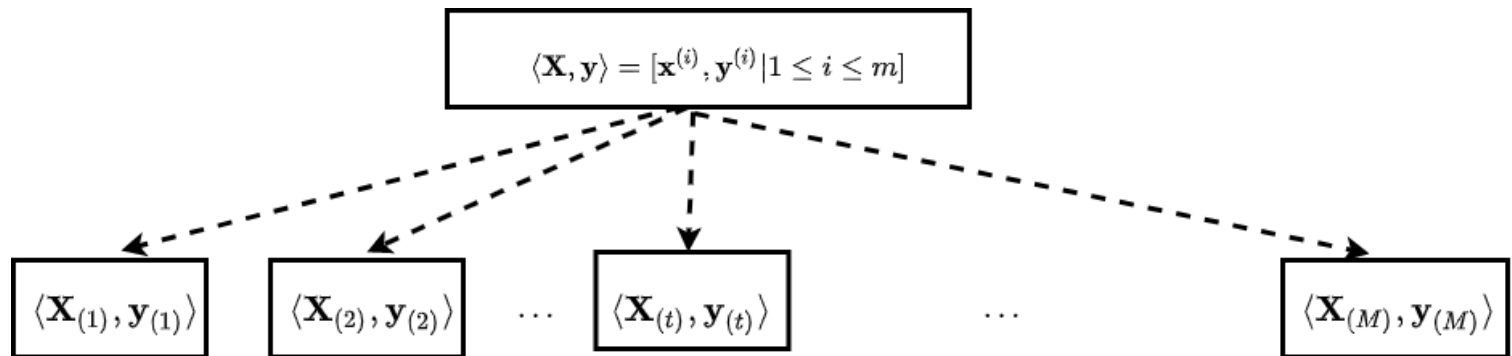
- Rather than create weak learners independently, i.e., a *set*
- Boosting creates a *sequence* of weak learners:  $M_{(0)}, M_{(1)}, \dots, M_{(M)}$
- Where the  $(t + 1)^{th}$  individual model in the sequence
- Focuses on correctly predicting those examples *incorrectly* predicted by the  $t^{th}$  individual model

## Notation

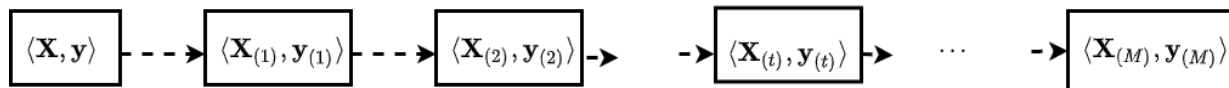
We will be dealing with many sequences. We use subscripts in parentheses to index elements of a sequence.

$$M_{(0)}, M_{(1)}, \dots, M_{(M)}$$

## Bagging



## Boosting



How do we get an individual model to focus on some particular examples ?

- By assigning each example a weight
- Increasing the probability that more heavily weighted examples are included in the training examples for the model

Let  $\text{say}_{(t)}^{(i)}$  denote the weight assigned to example  $i$  in the training set for the  $t^{\text{th}}$  individual model

The "say" is adjusted from the  $t^{\text{th}}$  model to the  $(t + 1)^{\text{th}}$  individual model

If example  $i$  is incorrectly predicted in model  $t$  :  $\text{say}_{(t+1)}^{(i)} > \text{say}_{(t)}^{(i)}$

If example  $i$  is correctly predicted in model  $t$  :  $\text{say}_{(t+1)}^{(i)} < \text{say}_{(t)}^{(i)}$

When bootstrapping, rather than drawing examples with equal probability

- Draw examples for model  $(t + 1)$  in proportion to its  $\text{say}_{(t+1)}^{(i)}$
- So examples that were "problematic" in model  $t$  are over-represented in training model  $(t + 1)$



- Boosting creates a collection of "specialists" (focus on hard to predict examples)
- Bagging creates a collection of "generalists", each a little better than random

# AdaBoost

AdaBoost is a particular model that uses boosting

- The individual models are Decision Trees
  - Usually depth 1; "stumps"
- There is an "importance" associated with each individual model
- Models with higher weight have a greater impact on ensemble prediction

Let

$\text{importance}_{(t)}$

denote the weight of the  $t^{th}$  individual model in the sequence.

- $\text{importance}_{(t)}$  is determined by the Performance Metric (e.g., Accuracy) of individual model  $t$
- The class predicted by the ensemble is the one with highest *importance-weighted* average (across individual models) probability

$$\hat{\mathbf{y}}^{(i)} = \underset{c}{\operatorname{argmax}} \sum_{t=1}^M (p_{(t),c}^{(i)} * \text{importance}_{(t)})$$

Thus, models that are more successful have greater weight.

# Gradient Boosting

Gradient Boosting is a "more mathematical" (less operational) approach to boosting

- A Loss Function is defined
- That measures the Loss  $\mathcal{L}_{(t)}$  of the ensemble consisting of the first  $t$  models in the sequence
- Computes the gradient of the Loss  $\mathcal{L}_{(t)}$
- Adds model  $(t + 1)$  to explicitly reduce the loss by moving in the direction of the gradient

$$\mathcal{L}_{(t+1)} < \mathcal{L}_{(t)}$$

We illustrate Gradient Boosting with a Regression task: predict continuous  $\hat{\mathbf{y}}$

- We will produce a sequence of models  $M_{(0)}, M_{(1)}, \dots, M_{(M)}$

We have not specified the functional form of the models  $M_t$

- It will typically be something like a Decision Tree and *not* Linear Regression

Model  $t$  will have

- Target denoted by  $e_{(t)}$
- Predictions denoted by  $\hat{e}_{(t)}$
- We define  $\hat{e}_{(0)} = \bar{\mathbf{y}}$ 
  - where  $\bar{\mathbf{y}}$  is the mean (over the  $m$  examples in training) of the target

$$\bar{\mathbf{y}} = \frac{1}{m} \sum_{i=1}^m \mathbf{y}^{(i)}$$

The prediction of the *ensemble* (rather than individual models) consisting of the first  $t$  models is

$$\hat{\mathbf{y}}_{(t)} = \sum_{t'=0}^t \alpha * \hat{\mathbf{e}}_{(t')}$$

That is: the *ensemble prediction* is the weighted sum of the *predictions of the individual models*.

- Unlike AdaBoost: the weights for each model are identical ( $\alpha$ )

## Loss function

The Loss function for the ensemble consisting of the first  $t$  models will be the MSE:

$$\mathcal{L}_{(t)} = \frac{1}{m} \sum_{i=1}^m (\mathbf{y}^{(i)} - \hat{\mathbf{y}}_{(t)}^{(i)})^2$$



The ensemble prediction  $\hat{\mathbf{y}}_{(t)}$  differs from the regression target  $\mathbf{y}$  by

$$\mathbf{y} - \hat{\mathbf{y}}_{(t)} = \mathbf{e}_{(t+1)}$$

We therefore set the target for model  $(t + 1)$

- To be  $\mathbf{e}_{(t+1)}$
- Which is the *residual* (error) of the target with respect to the ensemble prediction up to step  $t$

In other words: model  $(t + 1)$  is tasked with predicting the residual remaining after the ensemble prediction of the first  $t$  individual models.

To be clear, let's suppose the  $M_{(t)}$  are Decision Trees.

- Each tree  $M_{(t)}$  is constructed from *scratch*
  - It does not "extend" tree  $M_{(t-1)}$
- It's thus possible that two trees in the sequence have the same test
- Each tree  $M_{(t)}$  has a different target
  - The target for  $M_{(t)}$  is the remaining error between target  $\mathbf{y}$  and the prediction of the ensemble prefix of length  $(t - 1)$

- Because we defined  $\mathbf{e}_{(0)} = \bar{\mathbf{y}}$ 
  - $\mathbf{e}_{(1)} = \mathbf{y} - \bar{\mathbf{y}}$
- So model  $M_{(1)}$  is trying to predict the residual with respect to a simpler model (one that always predicts  $\bar{\mathbf{y}}$ )
- The ensemble of length 1 predicts
$$\bar{\mathbf{y}} + \hat{e}_{(1)}$$

Each model  $t$  in the sequence attempts to reduce the residual left over from the ensemble prediction of the prefix of length  $(t - 1)$ .

## Where are the gradients in Gradient Boosting ?

Consider the derivative of the Loss function (MSE) with respect to the ensemble prediction

$$\begin{aligned}\frac{\partial \mathcal{L}_{(t)}}{\partial \hat{\mathbf{y}}_t} &= \frac{\partial \frac{1}{m} \sum_{i=1}^m (\mathbf{y}^{(i)} - \hat{\mathbf{y}}_{(t)})^{(i)2}}{\partial \hat{\mathbf{y}}_t} \\ &= \frac{2}{m} (\mathbf{y}^{(i)} - \hat{\mathbf{y}}_{(t)}^{(i)}) * -1 \quad \text{chain rule} \\ &= -\frac{2}{m} \mathbf{e}_{(t+1)} \quad \text{definition of } \mathbf{e}_{(t+1)}\end{aligned}$$

That is: the gradient is proportional to the residual of the target with respect to the the prediction of the ensemble consisting of the first  $t$  models

You shouldn't be surprised to see the residual in the gradient; this is just a fact of the MSE

- It's derivative is closely related to the residual
- One can argue that the MSE was *chosen* exactly because of this property

So the ensemble of  $(t + 1)$  models can decrease the loss compared to the ensemble with  $t$  models

$$\mathcal{L}_{(t+1)} < \mathcal{L}_{(t)}$$

by making  $\hat{\mathbf{y}}_{(t+1)}$  equal to  $\hat{\mathbf{y}}_{(t)}$  plus the approximation of the residual.

This process of

- Minimizing a Loss function
- By incrementally updating predictions
  - In the direction (opposite direction really, because gradient is negative) the gradient is called *Gradient Descent*.

## Gradient Descent

- Will be our prime method of solving optimization problems, such as training models (minimizing Loss)
- Is a key component of Deep Learning

We will explore Gradient Descent in a subsequent module.



## Aside

Even though AdaBoost was created prior to Gradient Boosting

- It can be shown to be equivalent to Gradient Boosting when the Loss function is Exponential Loss.

In [12]: `print("Done")`

Done