# 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. Models differ by
    - different parameters/hyper-parameters:
        - Decision Trees of different depths or different features
        - Regression with polynomial features of different degrees
    - training datasets -subsets of full dataset

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
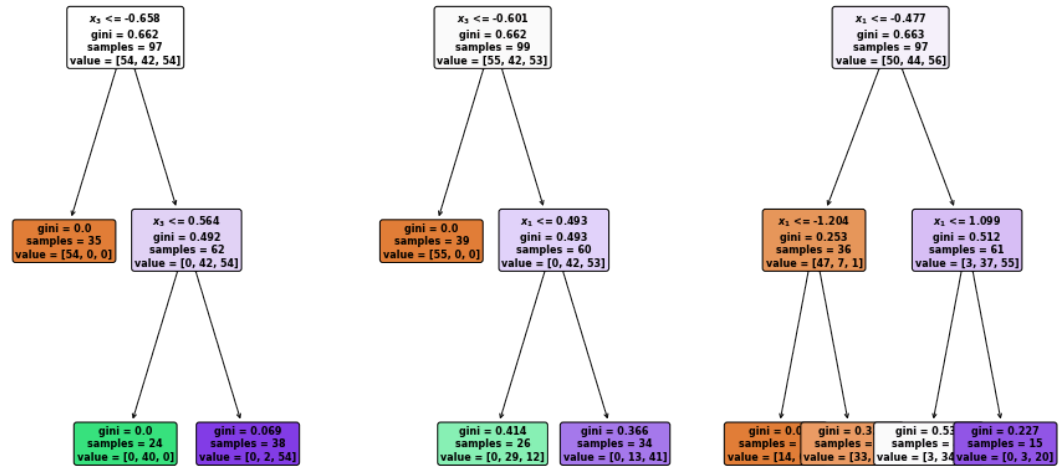- Makes them more robust to outliers

We will shortly explain how the subsets are chosen.

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

- trained on different subsets of the training dataset
  - details to follow: Bagging, Boosting
- we have limited the features used to $\mathbf{x}_1$, $\mathbf{x}_3$ only to make the diagrams smaller

In [7]: fig_ens

Out[7]:



| $x_3 <= -0.658$ | | $x_3 <= -0.601$ | | $x_1 <= -0.477$ | |
| gini = 0.662 | | gini = 0.662 | | gini = 0.663 | |
| samples = 97 | | samples = 99 | | samples = 97 | |
| value = [54, 42, 54] | | value = [55, 42, 53] | | value = [50, 44, 56] | |

**Tree 1:**
- gini = 0.0, samples = 35, value = [54, 0, 0]
- $x_3 <= 0.564$, gini = 0.492, samples = 62, value = [0, 42, 54]
  - gini = 0.0, samples = 24, value = [0, 40, 0]
  - gini = 0.069, samples = 38, value = [0, 2, 54]

**Tree 2:**
- gini = 0.0, samples = 39, value = [55, 0, 0]
- $x_1 <= 0.493$, gini = 0.493, samples = 60, value = [0, 42, 53]
  - gini = 0.414, samples = 26, value = [0, 29, 12]
  - gini = 0.366, samples = 34, value = [0, 13, 41]

**Tree 3:**
- $x_1 <= -1.204$, gini = 0.253, samples = 36, value = [47, 7, 1]
  - gini = 0.0, samples = , value = [14,
  - gini = 0.3, samples = , value = [33,
- $x_1 <= 1.099$, gini = 0.512, samples = 61, value = [3, 37, 55]
  - gini = 0.53, samples = , value = [3, 34
  - gini = 0.227, samples = 15, value = [0, 3, 20]

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 $\hat{\mathbf{y}}^{(\mathbf{i})}$ of example $i$, given features $\mathbf{x}^{(\mathbf{i})}$.

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

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

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

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

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

- the boundaries of each model differ
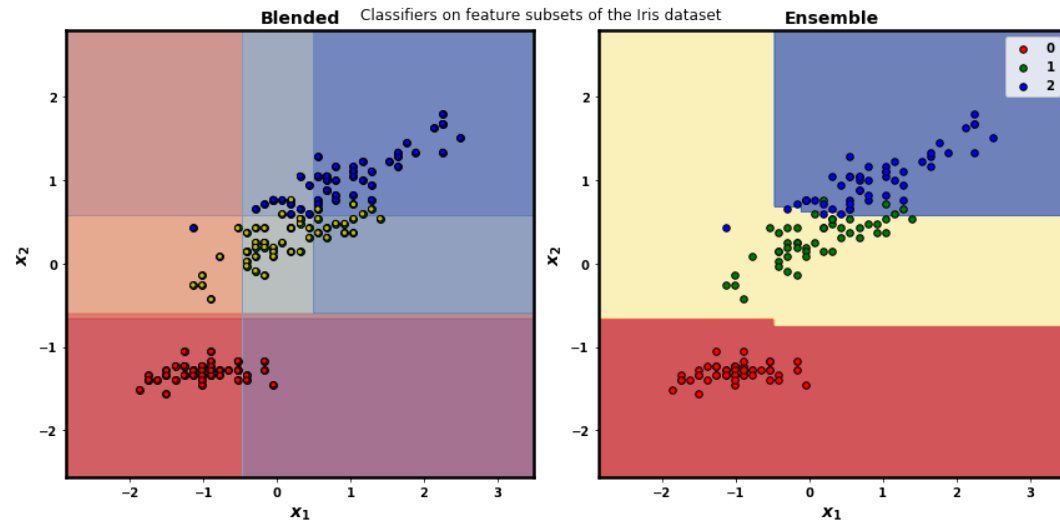- because they have been trained on different subsets of the full training dataset

`fig_submodels`

Classifiers on feature subsets of the Iris dataset
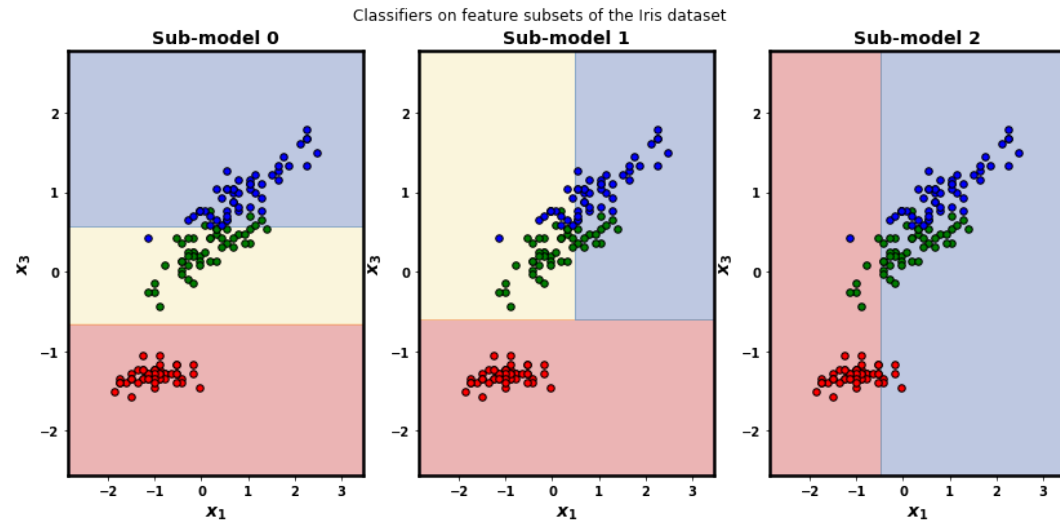
By superimposing these boundaries on top of one another, we can visualize the "vote"

Out[9]:



Classifiers on feature subsets of the Iris dataset

- 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
$$\langle \mathbf{X}_{(t)}, \mathbf{y}_{(t)} \rangle$$
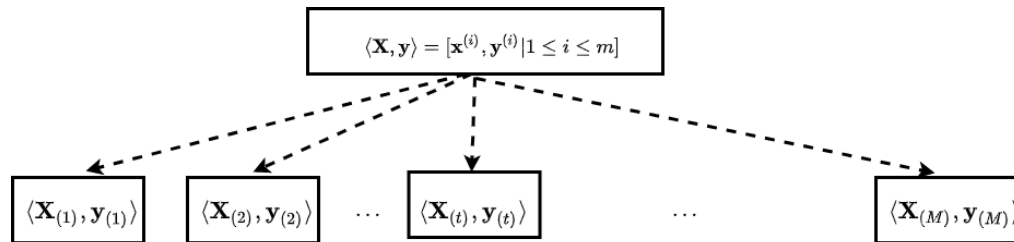on which to train the $t^{th}$ individual model

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* (or *bagging*) and results in

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

**Bagging**

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

$\langle \mathbf{X}_{(1)}, \mathbf{y}_{(1)} \rangle$   $\langle \mathbf{X}_{(2)}, \mathbf{y}_{(2)} \rangle$   $\cdots$   $\langle \mathbf{X}_{(t)}, \mathbf{y}_{(t)} \rangle$   $\cdots$   $\langle \mathbf{X}_{(M)}, \mathbf{y}_{(M)} \rangle$

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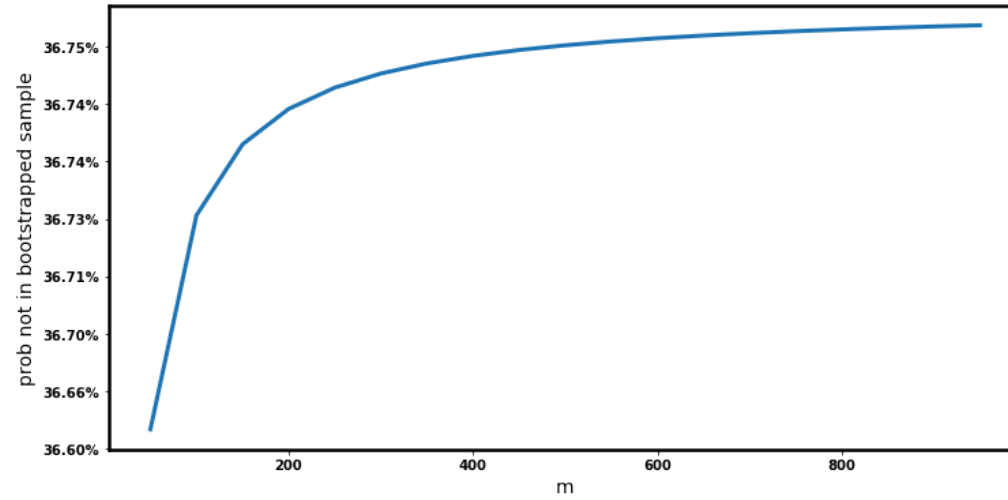
$$(1 - \frac{1}{m})^m$$

```
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: $M_{(0)}, M_{(1)}, \ldots, M_{(M)}$
- Where the $(t+1)^{th}$ individual model in the sequence
- Focuses on correctly predicting those examples *incorrectly* predicted by the $t^{th}$ individual model
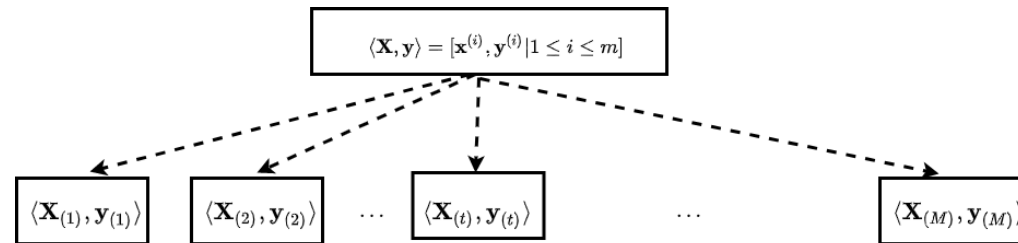
**Notation**

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

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

Recall:

- when bootstrapping/bagging
- each individual training dataset is drawn simultaneously from the full training dataset

**Bagging**

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

$$\langle \mathbf{X}_{(1)}, \mathbf{y}_{(1)} \rangle \quad \langle \mathbf{X}_{(2)}, \mathbf{y}_{(2)} \rangle \quad \cdots \quad \langle \mathbf{X}_{(t)}, \mathbf{y}_{(t)} \rangle \quad \cdots \quad \langle \mathbf{X}_{(M)}, \mathbf{y}_{(M)} \rangle$$

In contrast

- *boosting* creates the individual training datasets **sequentially**
- subset $\langle \mathbf{X}_{(t+1)}, \mathbf{y}_{(t+1)} \rangle$ for model $M_{(t+1)}$
- is chosen to compensate for the errors of **all prior** models $\{M_{(t')} \mid t' < t\}$

**Boosting**

$\langle \mathbf{X}, \mathbf{y} \rangle$ - - - ▶ $\langle \mathbf{X}_{(1)}, \mathbf{y}_{(1)} \rangle$ - - - ▶ $\langle \mathbf{X}_{(2)}, \mathbf{y}_{(2)} \rangle$ -▶ - ▶ $\langle \mathbf{X}_{(t)}, \mathbf{y}_{(t)} \rangle$ -▶ ⋯ - ▶ $\langle \mathbf{X}_{(M)}, \mathbf{y}_{(M)} \rangle$

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 $t^{th}$ individual model

The "say" is adjusted from the $t^{th}$ model to the $(t+1)^{th}$ 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 it's 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
$$\hat{\mathbf{y}}^{(\mathbf{i})} = \underset{c}{\operatorname{argmax}} \sum_{t=1}^{M} (p_{(t),c}^{(\mathbf{i})} * \text{importance}_{(t)})$$

Thus, models that are more successful have greater weight.

# Example: Boosting for a Regression task

Boosting is often associate with Classification tasks

- for example: the individual models are Decision Trees

Here we show how it may be used for Regression.

We will create a sequence of models $M_{(0)}, M_{(1)}, \ldots, M_{(M)}$

Each model is a Linear Regression.

Model $t$

$$
\begin{aligned}
\mathbf{e}_{(t)} &= \Theta_{(t)} \cdot \mathbf{x} + \epsilon_{(t)} \\
&= \hat{\mathbf{e}}_{(t)} + \epsilon_{(t)}
\end{aligned}
$$

where

- $\Theta_{(t)}$ are the parameters of the model
- $\mathbf{e}_{(t)}$ is the target (to be defined)
- $\hat{\mathbf{e}}_{(t)}$ is the predicted value
- $\epsilon_{(t)}$ is the prediction error

For the first model Model $M_{(0)}$

- target is $\mathbf{y}$
$$\mathbf{e}_{(0)} = \mathbf{y}$$
- we ignore the features $\mathbf{x}$ and predict the average
$$\hat{\mathbf{e}}_{(0)} = \bar{\mathbf{y}}$$
- the intercept parameters is $\bar{\mathbf{y}}$, all other parameters are $0$
$$\Theta_{(0)} = (\bar{\mathbf{y}}, 0, \ldots, 0)$$

For subsequent models $t + 1$

- the target $\mathbf{e}_{(t+1)}$
- is the *error* of the previous model $M_{(t)}$
$$\mathbf{e}_{(t+1)} = \epsilon_{(t)}$$

That is

- the goal of model $M_{(t+1)}$
- is to *reduce the error* of the previous model

| # | $\mathbf{e}_{(t)}$ | $\hat{\mathbf{e}}_{(t)}$ | $\epsilon_{(t)}$ |
|---|---|---|---|
| 0 | $\mathbf{y}$ | $\Theta_{(0)} \cdot \mathbf{x}$ | $\epsilon_{(0)}$ |
| 1 | $\epsilon_{(0)}$ | $\Theta_{(1)} \cdot \mathbf{x}$ | $\epsilon_{(1)}$ |
| $\vdots$ | | | |
| $M$ | $\epsilon_{(M)}$ | $\Theta_{(M)} \cdot \mathbf{x}$ | $\epsilon_{(M)}$ |

The predictions of the individual models in the ensemble consisting of the first $t$ models $M_{(0)}, \ldots, M_{(t)}$

- is combined into the *ensemble prediction* $\hat{\mathbf{y}}_{(t)}$
- as the weighted sum of the predictions of the individual models in the ensemble

$$
\begin{aligned}
\hat{\mathbf{y}}_{(t)} &= \sum_{t'=0}^{t} \alpha_{(t)} * \hat{\mathbf{e}}_{(t)} \\
&= \mathbf{x} \cdot \sum_{t'=0}^{t} \alpha_{(t)} * \Theta_{(t')}
\end{aligned}
$$

The boosting solution just derives the coefficients of a direct linear regression

$$
\hat{\mathbf{y}} = \Theta^* \cdot \mathbf{x}
$$

iteratively, as a sum

$$
\Theta^* = \sum_{t'=0}^{t} \alpha_{(t)} * \Theta_{(t')}
$$

We can show this another way

- the final model's error $\epsilon_{(M)}$
- is the error of a direct linear regression

Since
$$\hat{\mathbf{e}}_{(t)} = \mathbf{e}_{(t)} - \epsilon_{(t)}$$
we can write the ensemble prediction of all $M$ models as

$$
\begin{aligned}
\hat{\mathbf{y}}_{(M)} \quad &= \quad \sum_{t'=0}^{M} \left( \mathbf{e}_{(t')} - \epsilon_{(t')} \right) && \text{letting } \alpha_{(t)} = 1 \text{ for all } t \\
&= \quad \left( \mathbf{e}_{(0)} - \epsilon_{(0)} \right) + \sum_{t'=1}^{M} \left( \mathbf{e}_{(t')} - \epsilon_{(t')} \right) && \text{moving } t' = 0 \text{ out of the sum} \\
&= \quad \left( \mathbf{e}_{(0)} - \epsilon_{(0)} \right) + \sum_{t'=1}^{M} \left( \epsilon_{(t'-1)} - \epsilon_{(t')} \right) && \text{since } \mathbf{e}_{(t')} = \epsilon_{(t'-1)} \\
&= \quad \mathbf{e}_{(0)} - \epsilon_{(M)} && \text{the negative } -\epsilon_{(t)} \text{ term occur} \\
&&& \text{is canceled by the positive } \epsilon_{(t'-} \\
&&& \text{e.g., } -\epsilon_{(0)} \text{ (outside the sum) i} \\
&&& \text{and } -\epsilon_{(t)} \text{ term occurring wh} \\
&= \quad \mathbf{y} - \epsilon_{(M)} && \text{since } \mathbf{e}_{(0)} = \mathbf{y} \\
\mathbf{y} \quad &= \quad \hat{\mathbf{y}}_{(M)} + \epsilon_{(M)} && \text{re-arranging terms}
\end{aligned}
$$

# Example: Boosting for a Classification task

Although we won't construct an example for Classification, there are some important points to consider.

Each model is created from *scratch*

- Model $M_{(t+1)}$ does **not** extend model $M_{(t)}$
- For example, if the models are Decision Trees
    - the tree for $M_{(t+1)}$ is not an expansion of the tree for $M_{(t)}$

Although the models are created *independently*

- their *training datasets* are constructed sequentially

# Gradient Boosting

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

We associate a *Loss Function*

$$\mathcal{L}_{(t)}$$

- with the *ensemble prediction* $\hat{\mathbf{y}}_{(t)}$
- of the first $t$ models: $M_{(0)}, M_{(1)}, \ldots, M_{(t)}$

Model $M_{(t+1)}$ is constructed so as to reduce the ensemble loss

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

In fact, our Regression example above used Gradient Boosting !

Define the Loss for the ensemble consisting of the first $t$ models as the MSE error

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

where

$$\hat{\mathbf{y}}_{(t)}^{(i)}$$

is the ensemble prediction for training example $i$.

Compute the gradient

- of the Loss $\mathcal{L}_{(t)}$
- with respect to ensemble prediction $\hat{\mathbf{y}}_{(t)}$

$$\nabla_{(t)} = \frac{\partial \mathcal{L}_{(t)}}{\partial \hat{\mathbf{y}}_{(t)}}$$

By definition of the Gradient

- we can reduce the loss $\mathcal{L}_{(t+1)}$ of the next model
- by creating ensemble prediction $\hat{\mathbf{y}}_{(t+1)}$ as

$$\hat{\mathbf{y}}_{(t+1)} = \hat{\mathbf{y}}_{(t)} - \alpha * \nabla_{(t)}$$

The Gradient

- is in the direction of an *increase* of Loss
- so we adjust in the *negative* direction of the gradient
- in proportion to $\alpha$ (the *learning rate*)

The iterative process of reducing the Loss by moving in the direction of the gradient

- is called *Gradient Descent*
- and is the basis for the optimization used in Deep Learning

Referring back to our Regression example:

We can compute the gradient for the MSE Loss we have chosen:

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

So our ensemble predictor (for the larger ensemble of $t + 1$ models)

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

should increase our previous (smaller ensemble of $t$ models) ensemble prediction

$$\hat{\mathbf{y}}_{(t)}$$

proportional to the error

$$\epsilon_{(t)}$$

in order to reduce the Loss

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

The larger ensemble adds one more term to the summation expression for the ensemble predictor

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

Note that the target $\mathbf{e}_{(t+1)}$ for model $t+1$ is

$$\mathbf{e}_{(t+1)} = \epsilon_{(t)}$$

so

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

That is

- the term added to $\hat{\mathbf{y}}_{(t)}$ to obtain $\hat{\mathbf{y}}_{(t+1)}$
- is approximately equal to the Gradient of Loss
- as required by Gradient Descent

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

Done