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 model 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.

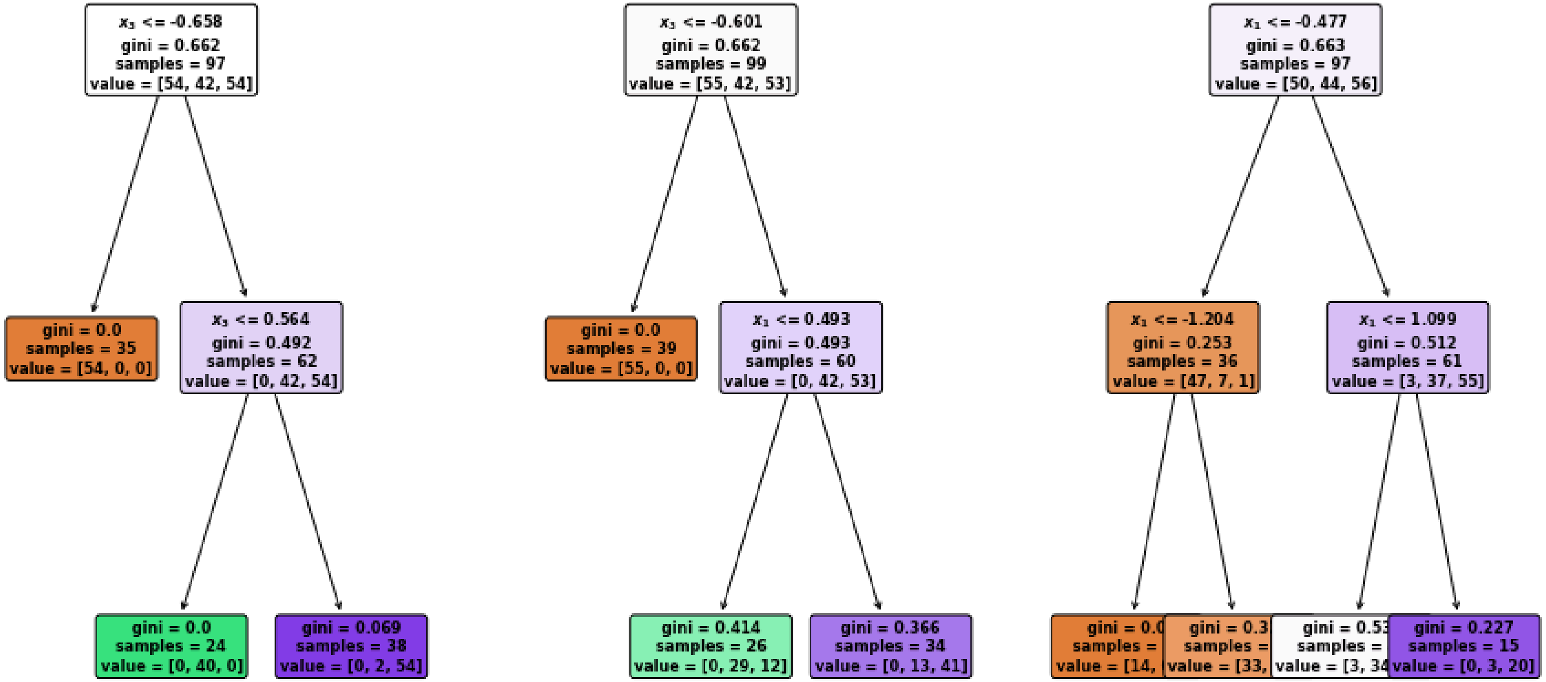
Here is an Ensemble of individual models of the same type: Decision Trees

- classification among 3 classes
- trained on different subsets of the training dataset
    - details to follow: Bagging, Boosting
- we have limited the features used to $\backslash\mathbf{x}_1, \backslash\mathbf{x}_3$ only to make the diagrams smaller

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 $m' \geq \ceil{M/2}$ 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 (and hence, $m' \geq \ceil{M/2}$) increases.
    - since $(1-A) < 1$
    - when raised to a power (m') the second term goes to 0

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 $\backslash\hat{\mathbf{y}}^{\backslash \mathrm{ip}}$ of example i, given features $\backslash\mathbf{x}^{\backslash \mathrm{ip}}$.

Let $p^{\backslash \mathrm{ip}}_{\backslash \mathrm{tp},c}$

- Denote the probability predicted by the $^{\mathtt{th}}$ individual classifier
- That target $\backslash\mathbf{y}^{\backslash \mathrm{ip}}$ is in category $c \in C$
- Given features $\backslash\mathbf{x}^{\backslash \mathrm{ip}}$

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

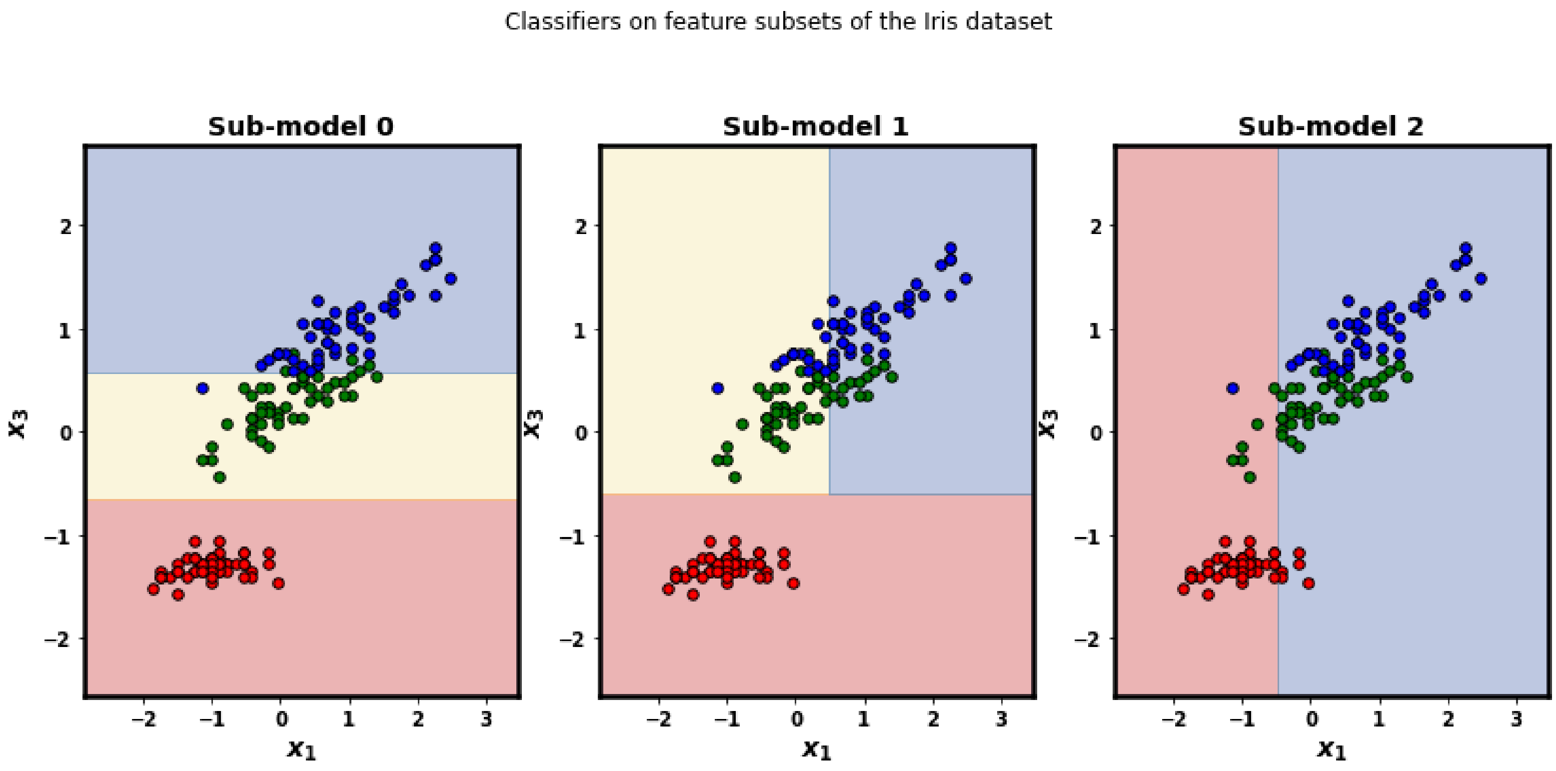
$$\backslash\hat{\mathbf{y}}^{\backslash \mathrm{ip}} = \backslash\mathrm{argmax}c \sum_{=1}^{M} p^{\backslash \mathrm{ip}}_{\backslash \mathrm{tp},c}$$

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

- 3 classes: red, green, blue
- the boundaries of each model differ
- because they have been trained on different subsets of the full training dataset

In [8]:
```
fig_submodels
```

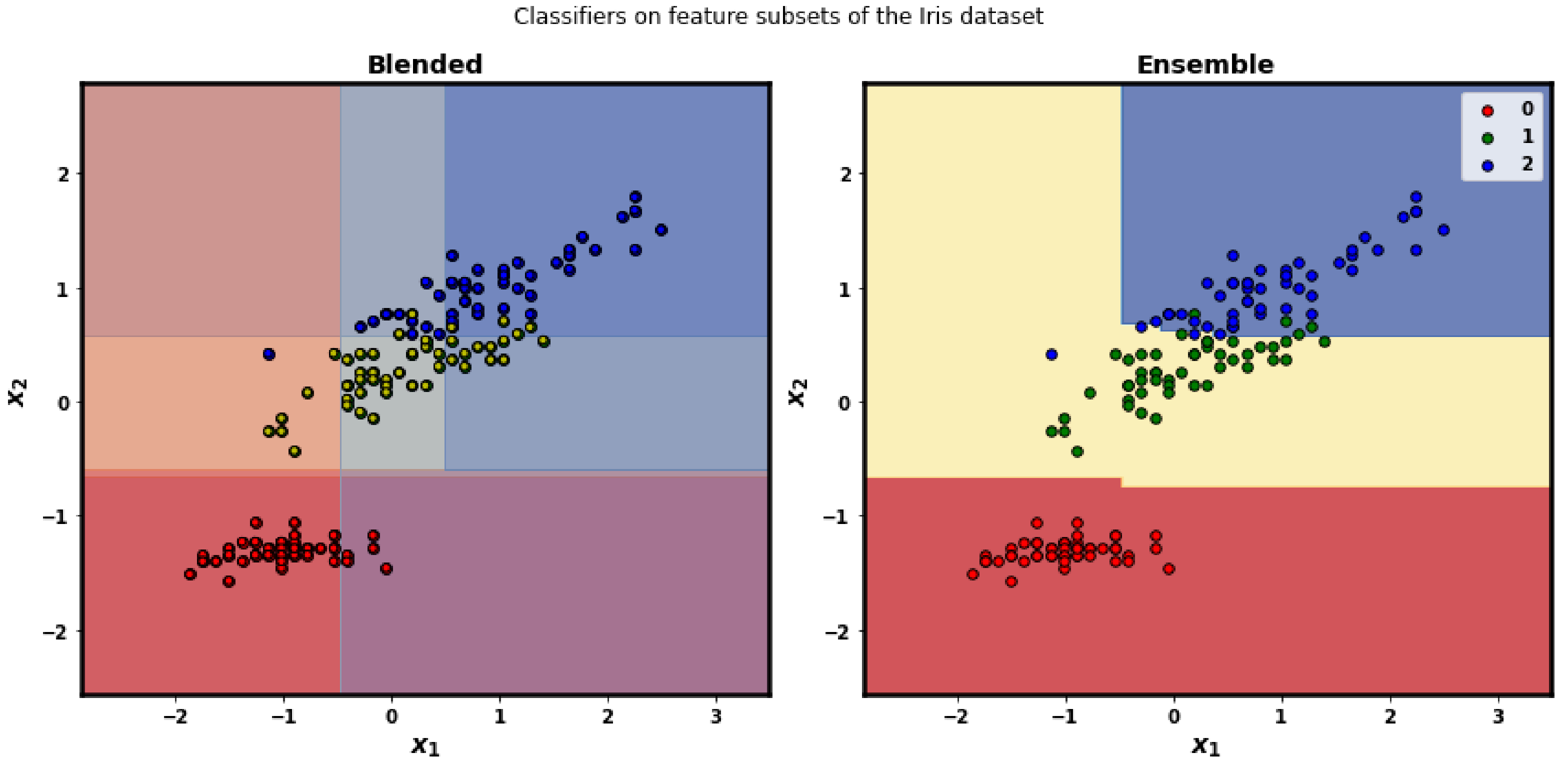
Out[8]:



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

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

we construct a restricted set of examples

$$\langle \backslash\mathbf{X}_{\backslash\mathrm{tp}}, \backslash\mathbf{y}_{\backslash\mathrm{tp}} \rangle$$

on which to train the `th` individual model

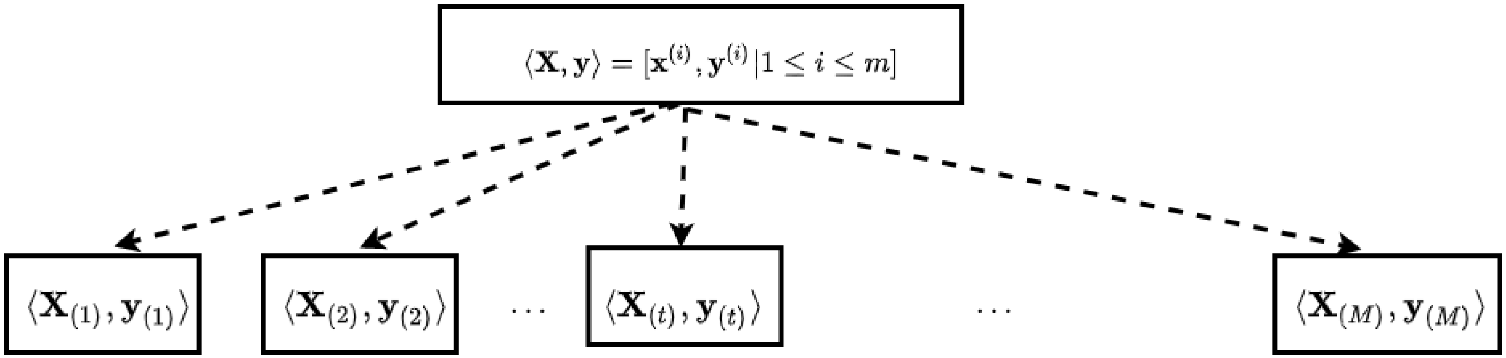
The restricted set is constructed by

- Selecting m examples at random from $\langle \backslash\mathbf{X}, \backslash\mathbf{y} \rangle$
- *With replacement*
- So it is possible for an example i' to appear more than once in $\langle \backslash\mathbf{X}_{\backslash \mathrm{tp}}, \backslash\mathbf{y}_{\backslash \mathrm{tp}} \rangle$

This process is called *bootstrapping* (or *bagging*) and results in

- $\langle \backslash\mathbf{X}_{\backslash\mathrm{tp}}, \backslash\mathbf{y}_{\backslash\mathrm{tp}} \rangle = [\backslash\mathbf{x}^{(i')}, \backslash\mathbf{y}^{(i')} | i' \in \{i_1, \ldots, i_m\}]$
- Where $i_1, \ldots, i_m$ are the indices of the m chosen examples

Bagging



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

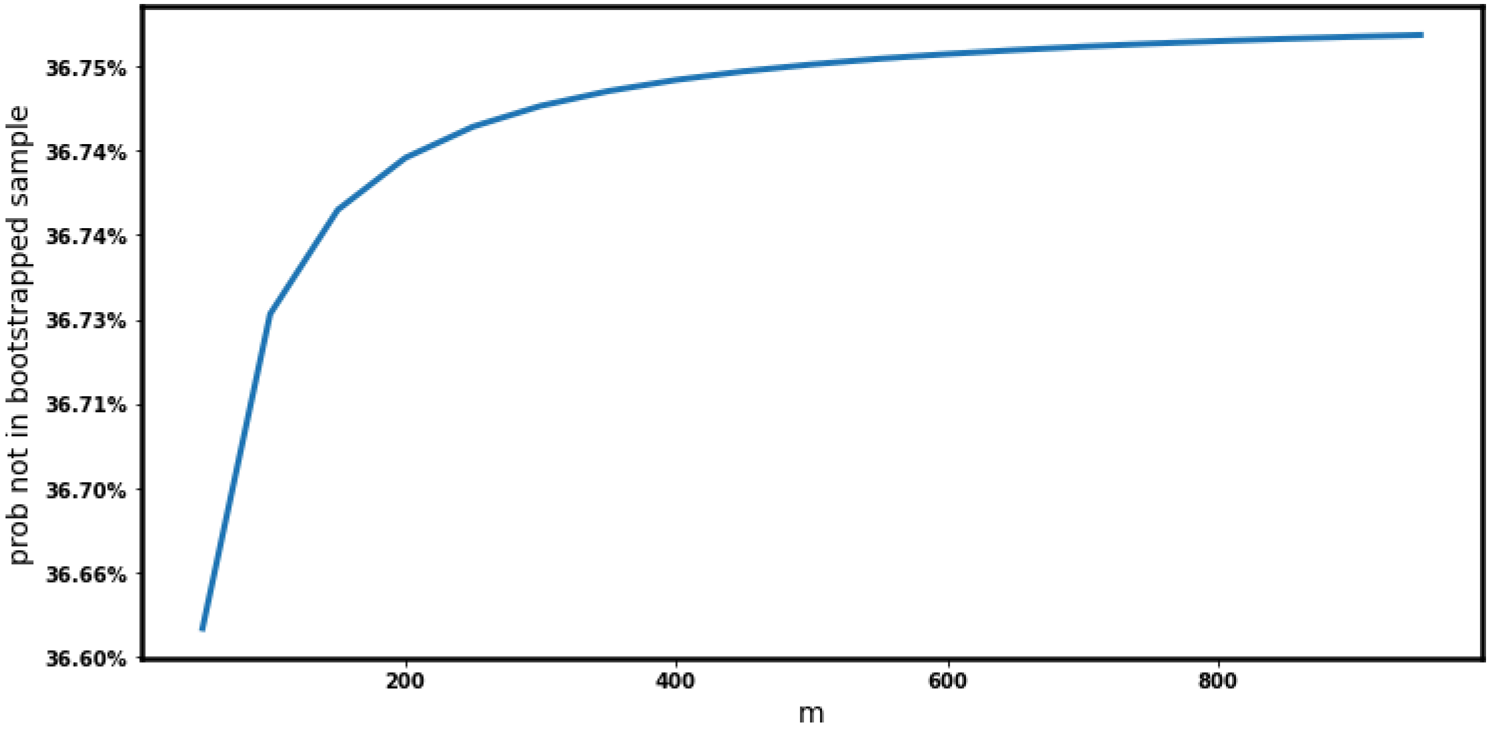
- The probability of a particular example i **not** being in $\langle \backslash\mathbf{X}_{\backslash\mathrm{tp}}, \backslash\mathbf{y}_{\backslash\mathrm{tp}} \rangle$ is
$$(1 - \frac{1}{m})^m$$

Let's plot this probability as a function of the training dataset size m

In [11]:

```
fig
```

Out[11]:



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

Why is this a potential advantage ?

- the model may perform better (in-sample) on duplicated examples
- the model 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 !

- a built-in validation dataset

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 $(+1)^{\texttt{th}}$ individual model in the sequence
- Focuses on correctly predicting those examples *incorrectly* predicted by the $^{\texttt{th}}$ individual model

Notation

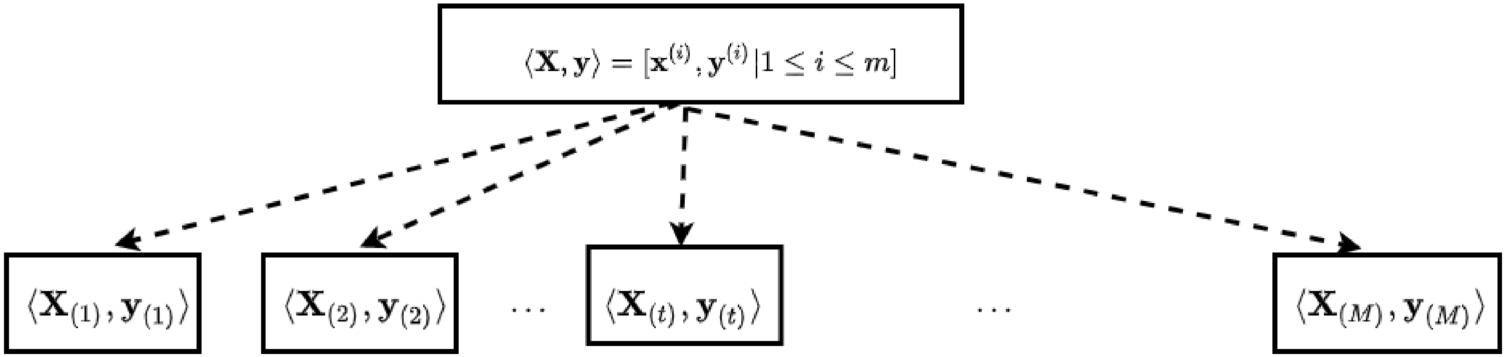
We will be dealing with many sequences. We use subscripts in parentheses to index elements of a sequence.

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

Recall:

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

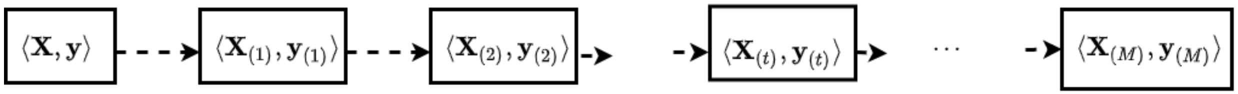
Bagging



In contrast

- *boosting* creates the individual training datasets **sequentially**
- subset $\langle \backslash\mathbf{X}_{(+1)}, \backslash\mathbf{y}_{(+1)} \rangle$ for model $M_{(+1)}$
- is chosen to compensate for the errors of **all prior** models $\{M_{(')} \mid ' <\}$

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
    - examples with poor predictions by earlier models are over-weighted in the subsequent model

Let $\mathrm{say}_{\backslash\mathrm{tp}}^{\backslash\mathrm{ip}}$ denote the weight assigned to example i in the training set for the $^{\texttt{th}}$ individual model

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

$$\begin{aligned}
&\text{If example } i \text{ is incorrectly predicted in model } : && \mathrm{say}_{(\texttt{+1})}^{\backslash\mathrm{ip}} > \mathrm{say}_{\backslash\mathrm{tp}}^{\backslash\mathrm{ip}} \\
&\text{If example } i \text{ is correctly predicted in model } : && \mathrm{say}_{(\texttt{+1})}^{\backslash\mathrm{ip}} < \mathrm{say}_{\backslash\mathrm{tp}}^{\backslash\mathrm{ip}}
\end{aligned}$$

When bootstrapping, rather than drawing examples with equal probability

- Draw examples for model $(+1)$ in proportion to it's $\mathrm{say}^{\backslash\mathrm{ip}}_{(+1)}$
- So examples that were "problematic" in model are over-represented in training model $(+1)$

- 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

$$\mathrm{importance}_{\backslash\mathrm{tp}}$$

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

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

$$\backslash\hat{\mathbf{y}}^{\backslash\mathrm{ip}} = \backslash\mathrm{argmax}c \sum_{=1}^{M} (p^{\backslash\mathrm{ip}}_{\backslash\mathrm{tp},c} * \mathrm{importance}_{\backslash\mathrm{tp}})$$

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.

Our goal is to solve for the optimal parameters Θ^* for the Linear Regression

$$\backslash\mathbf{y} = \Theta \cdot \backslash\mathbf{x} + \epsilon$$

That is, Θ^* minimizes the MSE

$$\Theta^* = \backslash\mathbf{argmin}\Theta \frac{1}{m} \sum_i (\backslash\mathbf{y}^{\backslash\mathrm{ip}} - \Theta \cdot \backslash\mathbf{x}^{\backslash\mathrm{ip}})^2)$$

The Boosting method

- creates a sequence of approximations of the optimal parameters Θ
$$\Theta^*_{(0)}, \Theta^*_{(1)}, \ldots$$
that approach the optimal Θ^*.

Boosting creates a sequence of models

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

such that model $M_{(+1)}$ compensates for the errors of earlier models $M_{(')}$ in the sequence.

Each model $M_{\backslash\mathrm{tp}}$ in the sequence has a functional form that is a Linear

Model

$$\begin{aligned}
\backslash\mathrm{e}_{\backslash\mathrm{tp}} &= \Theta_{\backslash\mathrm{tp}} \cdot \backslash\mathbf{x} &+& \quad \epsilon_{\backslash\mathrm{tp}} \\
&= \hat{\backslash}\mathrm{e}_{\backslash\mathrm{tp}} &+& \quad \epsilon_{\backslash\mathrm{tp}} \quad \text{define } \hat{\backslash}\mathrm{e}_{\backslash\mathrm{tp}} = \Theta_{\backslash\mathrm{tp}} \cdot \backslash\mathbf{x}
\end{aligned}$$

where

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

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

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

For subsequent models +1

- the target $\backslash\mathrm{e}_{(+1)}$
- is the *error* of the previous model $M_{\backslash\mathrm{tp}}$

$$\backslash\mathrm{e}_{(+1)} = \epsilon_{\backslash\mathrm{tp}}$$

Model $M_{(+1)}$ tries to find

- additional explanatory power in the $\backslash\mathbf{x}$
- by creating $\widehat{\backslash\mathrm{e}}_{(+1)} = \Theta_{(+1)} \cdot \backslash\mathbf{x}$
- that reduces the previous error $\epsilon_{\backslash\mathrm{tp}}$

If no additional explanatory power is possible, i.e., $\epsilon_{\backslash \mathrm{tp}}$ (the new target)

- is 0
- or uncorrelated with $\backslash \mathbf{x}$

then the sequence of models is not extended further.

Here is a picture of the sequence of models along with their

- targets $\backslash \mathbf{e}_{\backslash \mathrm{tp}}$
- predictions $\hat{\backslash \mathbf{e}}_{\backslash \mathrm{tp}}$
- and errors $\epsilon_{\backslash \mathrm{tp}}$

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

The prediction $\hat{\backslash\mathbf{y}}_{\backslash\mathrm{tp}}$ of the ensemble of the first models

- is a weighted sum of the predictions of the targets of these models

$$\begin{aligned}\hat{\backslash\mathbf{y}}_{\backslash\mathrm{tp}} &= \sum_{'=0}^{\alpha_{\backslash\mathrm{tp}} * \hat{\backslash\mathbf{e}}_{\backslash\mathrm{tp}}} \\ &= \backslash\mathbf{x} \cdot \sum_{'=0}^{\alpha_{\backslash\mathrm{tp}} * \boldsymbol{\Theta}_{(')}}\end{aligned}$$

The boosting solution thus derives the coefficients Θ^* of a direct Linear Regression model

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

iteratively, as a sum

$$\Theta^* = \sum_{'=0}^{\alpha_{\backslash\mathrm{tp}} * \boldsymbol{\Theta}_{(')}}$$

The parameter $\Theta_{\backslash\mathrm{tp}}$ at step

We will demonstrate a method

- called *Gradient Descent*
- that will define the sequence of parameter *updates* $\Theta_{(1)}, \ldots, \Theta_{(M)}$

in a subsequent module.

We call the new sequence of Θ's "updates"

- since $\Theta_{\backslash \mathrm{tp}}$
- updates the estimate of Θ^* from the shorter sequence of (-1) models.

Since these boosting process uses Gradients, it is called *Gradient Boosting*.

Note

If we had used Linear Regression to obtain the updates $\Theta_{\setminus \mathrm{tp}}$ for model $M_{\setminus \mathrm{tp}}$

- then $\epsilon_{\setminus \mathrm{tp}}$ is *uncorrelated* with $\setminus \mathbf{x}$
    - property of Ordinary Least Squares (OLS) Regression
- the sequence of models is not extended

We now relate the final prediction $\backslash\hat{\text{y}}_{(M)}$

- to the true target $\backslash\text{y}$.

Since the prediction $\backslash\hat{\text{e}}_{\backslash\text{tp}}$ of model $M_{\backslash\text{tp}}$ is the difference between target and error

$$\backslash\hat{\text{e}}_{\backslash\text{tp}} = \backslash\text{e}_{\backslash\text{tp}} - \epsilon_{\backslash\text{tp}}$$

we can write the ensemble prediction (sum across the predictions of all models) as

$$
\begin{array}{lll}
\backslash\hat{\text{y}}_{(M)} & = \sum_{'=0}^{M} \left(\backslash\text{e}_{(')} - \epsilon_{(')} \right) & \text{letting } \alpha_{\backslash\text{tp}} = 1 \text{ for all} \\
& = \left(\backslash\text{e}_{(0)} - \epsilon_{(0)} \right) + \sum_{'=1}^{M} \left(\backslash\text{e}_{(')} - \epsilon_{(')} \right) & \text{moving } ' \texttt{ = 0 out of the su} \\
& = \left(\backslash\text{e}_{(0)} - \epsilon_{(0)} \right) + \sum_{'=1}^{M} \left(\epsilon_{('-1)} - \epsilon_{(')} \right) & \text{since } \backslash\text{e}_{(')} = \epsilon_{('-1)} \\
& & \text{the target of model } ' \texttt{ is the} \\
& = \backslash\text{e}_{(0)} - \epsilon_{(M)} & \text{the negative } -\epsilon_{()} \text{ term occur} \\
& & \text{is canceled by the positive } \epsilon_{('-} \\
& & \text{e.g., } -\epsilon_{(0)} \text{ (outside the sum) i} \\
& & \text{and } -\epsilon_{()} \text{ term occurring whe} \\
& = \backslash\text{y} - \epsilon_{(M)} & \text{since } \backslash\text{e}_{(0)} = \backslash\text{y}
\end{array}
$$

That is, $\backslash\hat{\mathbf{y}}_{(M)}$

- is an approximation of true target $\backslash\mathbf{y}$
- with error $\epsilon_{(M)}$

Re-arranging terms we get the familiar linear form for Linear Regression

$$\backslash\mathbf{y} = \backslash\hat{\mathbf{y}}_{(M)} + \epsilon_{(M)}$$

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

Although the models are created *independently*

- their *training datasets* are constructed sequentially

In [12]:

```
print("Done")
```

```
Done
```