NYU CS-GY 6923 Machine Learning

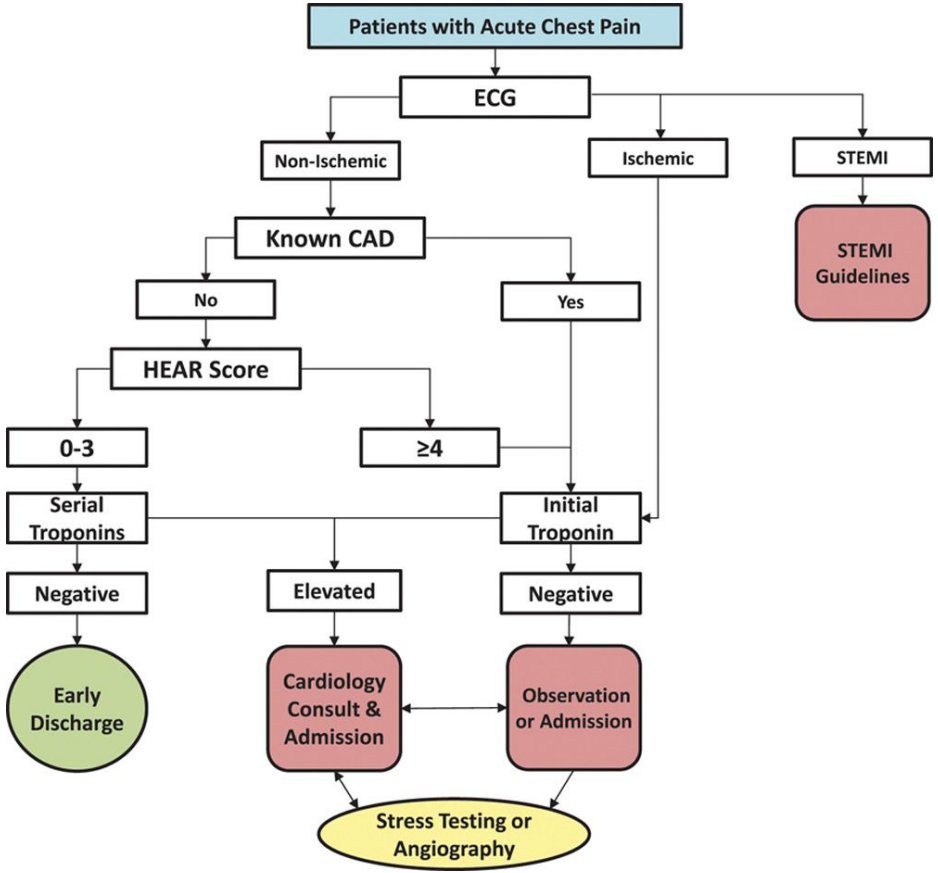
Prof. Pavel Izmailov

Today

- Decision Trees
- Ensembling, Bagging
- Random Forest
- Boosting
- Deep Ensembles


Decision Trees

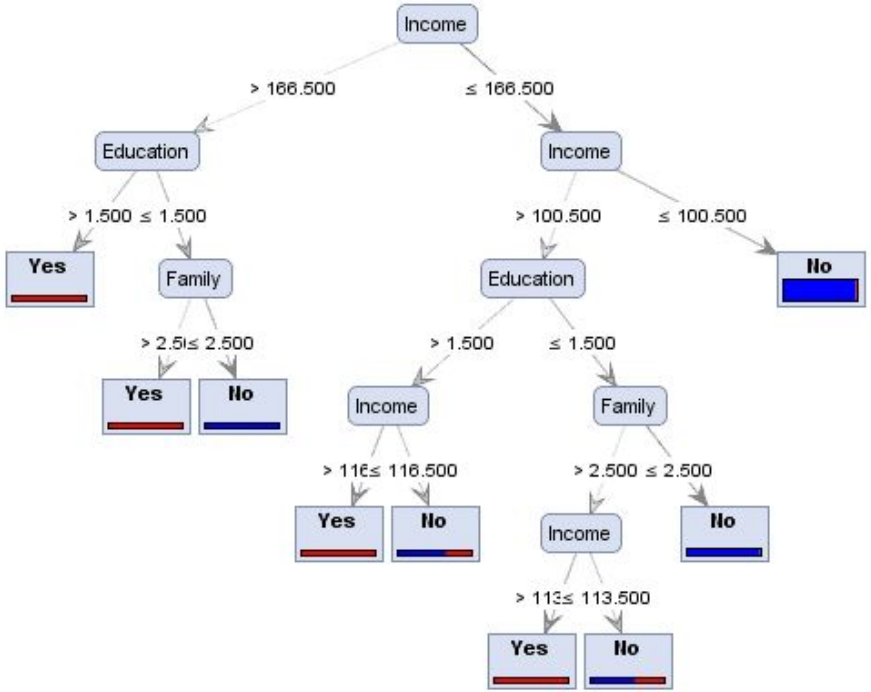
Decision Trees



Human decision making is often structured like a tree:

- Emergency room protocols

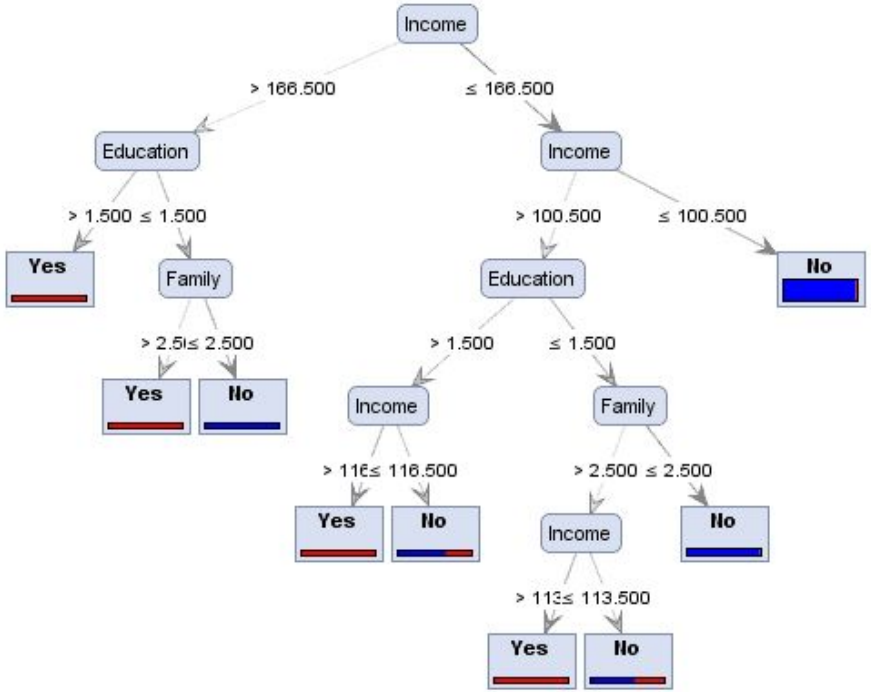
Decision Trees



Human decision making is often structured like a tree:

- Emergency room protocols
- Credit scoring

Decision Trees



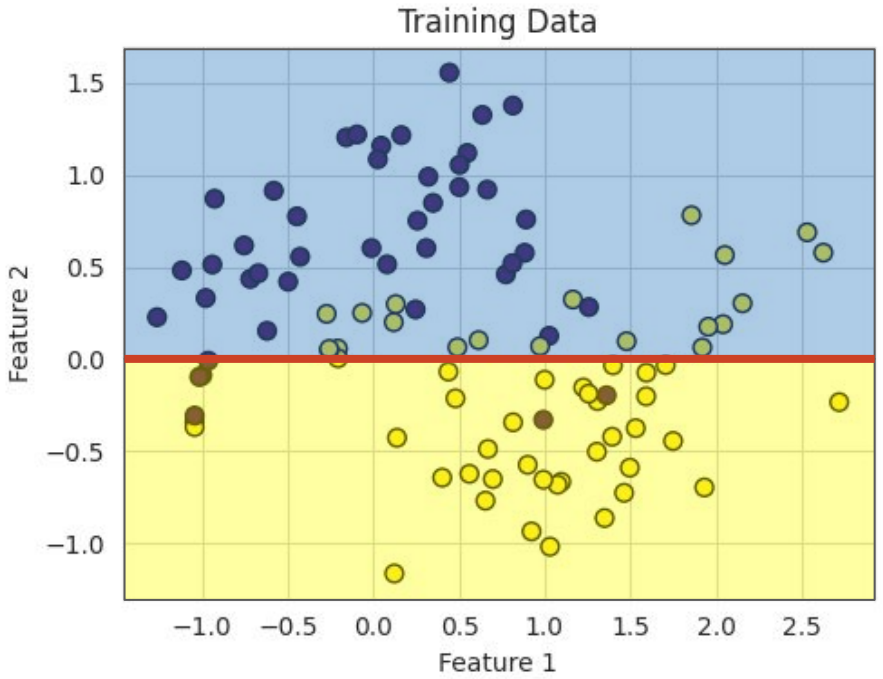
Human decision making is often structured like a tree:

- Emergency room protocols
- Credit scoring
- …

Trees are simple, interpretable, flexible, non-linear.

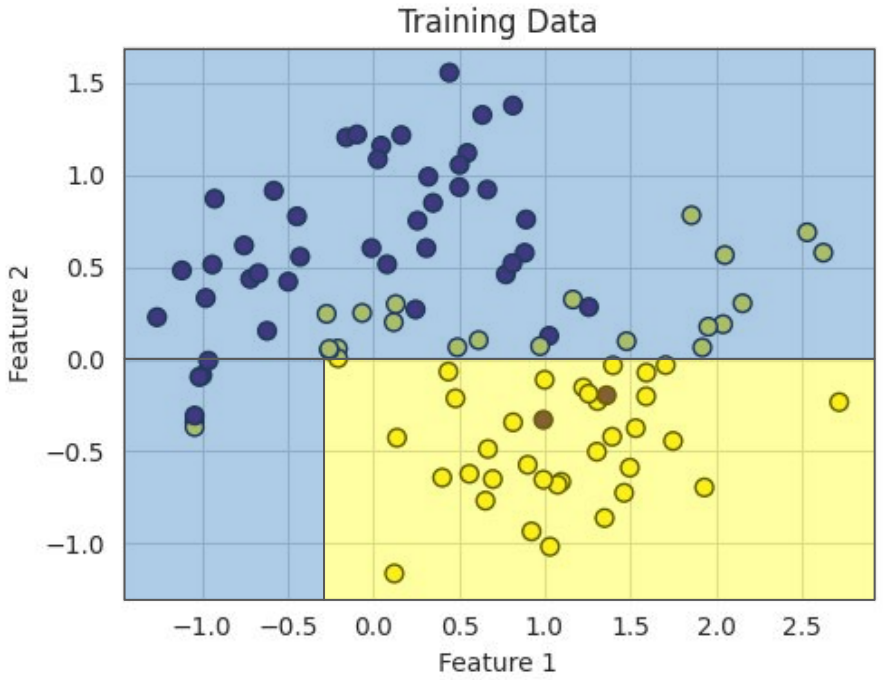
Decision Trees

- F2 < 0?
  - Yes ⇒ cls 0
  - No ⇒ cls 1



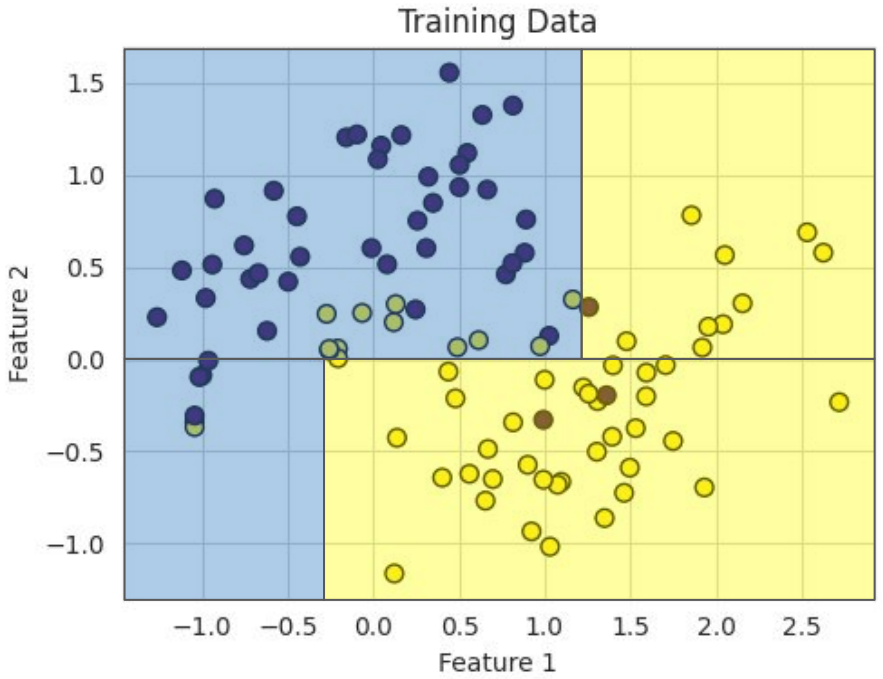
Decision Trees

- F2 < 0?
  - Yes ⇒ F1 < -0.25?
    - Yes ⇒cls 1
    - No ⇒ cls 0
  - No ⇒ cls 1



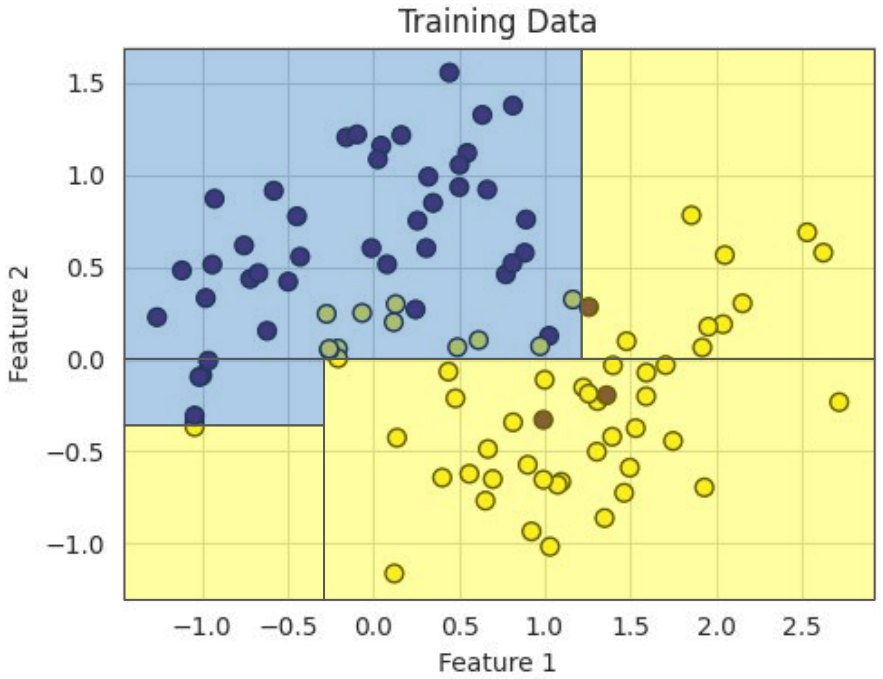
Decision Trees

- F2 < 0?
  - Yes ⇒ F1 < -0.25?
    - Yes ⇒cls 1
    - No ⇒ cls 0
  - No ⇒ F1 < 1.25?
    - Yes ⇒cls 1
    - No ⇒ cls 0



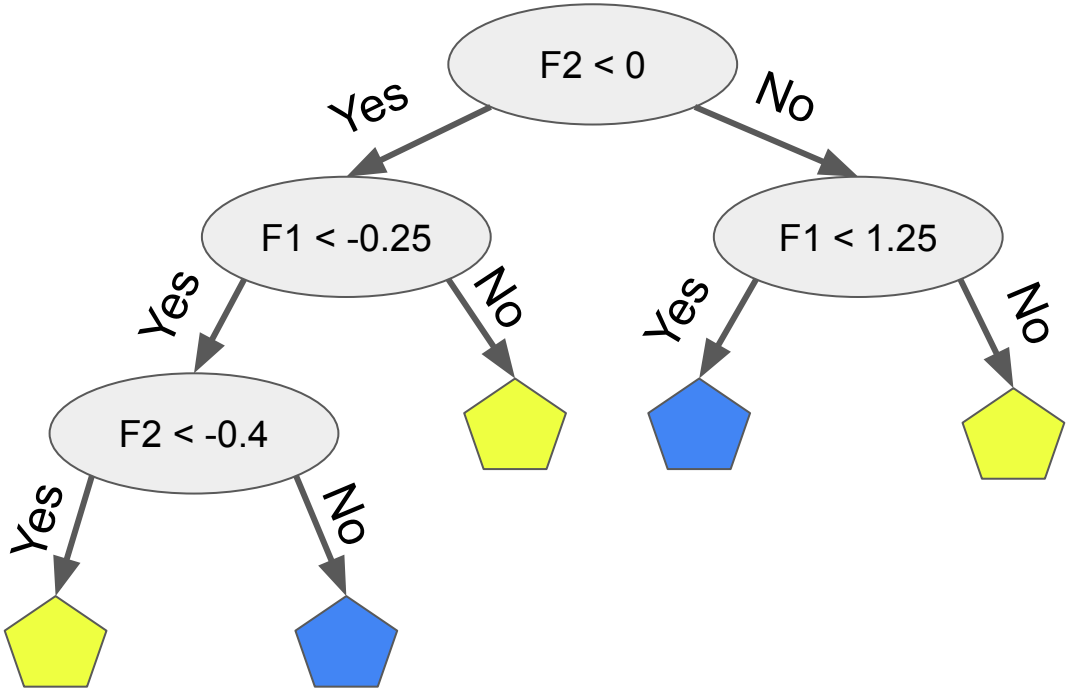
Decision Trees

- F2 < 0?
  - Yes ⇒ F1 < -0.25?
    - F2 < -0.4
      - Yes ⇒ cls 0
      - No ⇒ cls 1
    - No ⇒ cls 0
  - No ⇒ F1 < 1.25?
    - Yes ⇒cls 1
    - No ⇒ cls 0

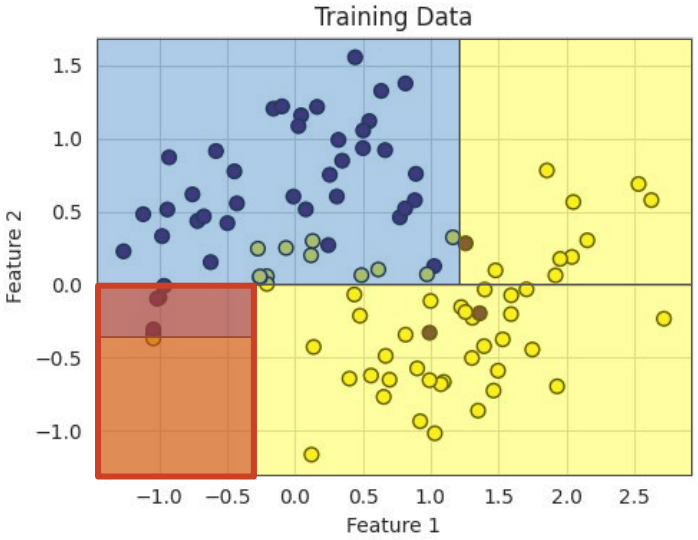


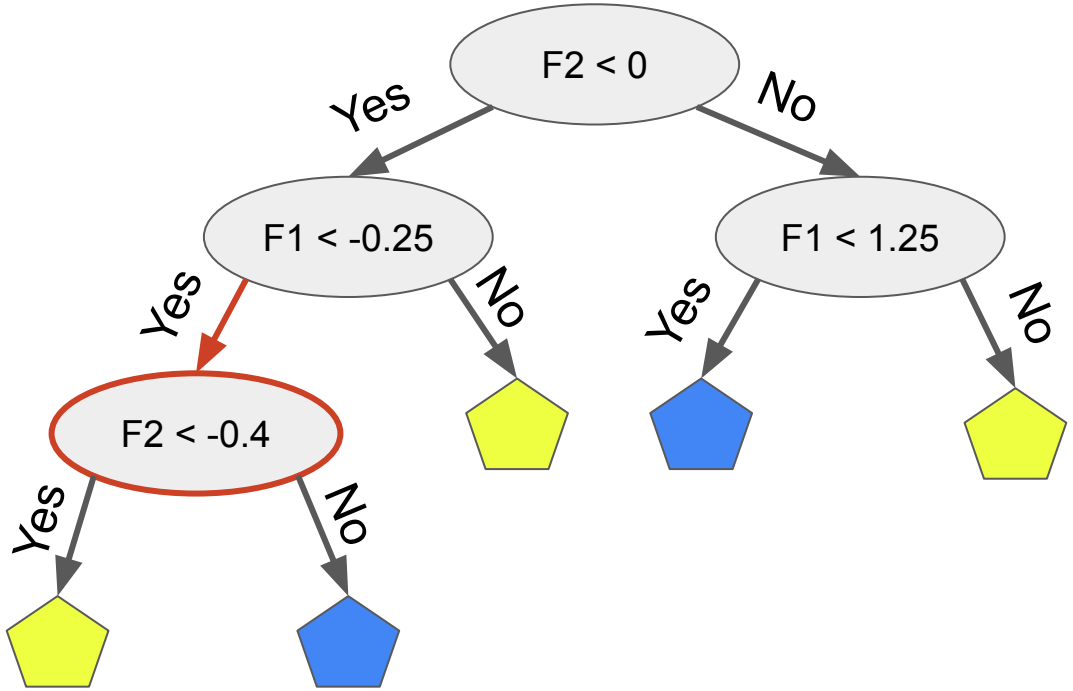
Decision Trees

- F2 < 0?
  - Yes ⇒ F1 < -0.25?
    - F2 < -0.4
      - Yes ⇒ cls 0
      - No ⇒ cls 1
    - No ⇒ cls 0
  - No ⇒ F1 < 1.25?
    - Yes ⇒cls 1
    - No ⇒ cls 0



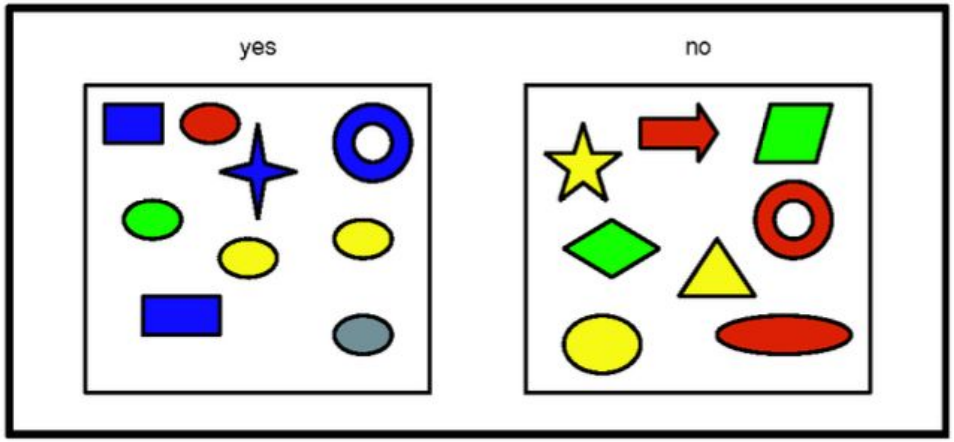
Decision Trees

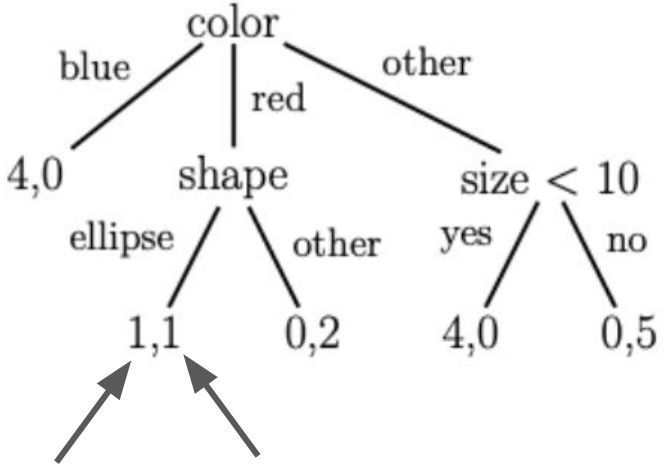




Each node corresponds to a rectangle in the feature space

Decision Trees





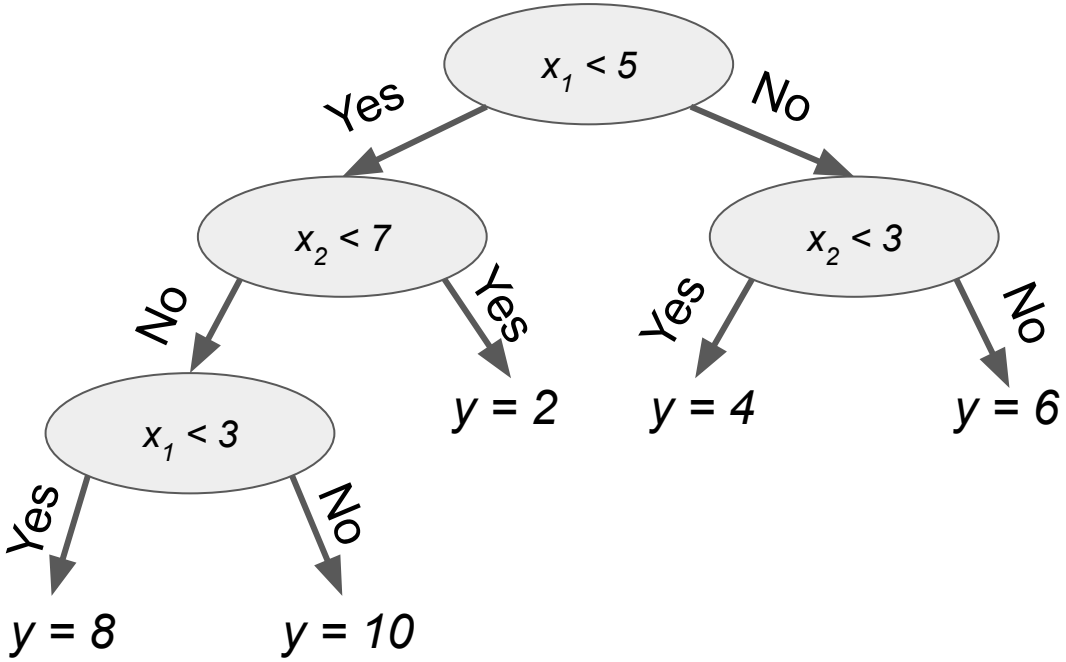
objects of class “yes” in this leaf

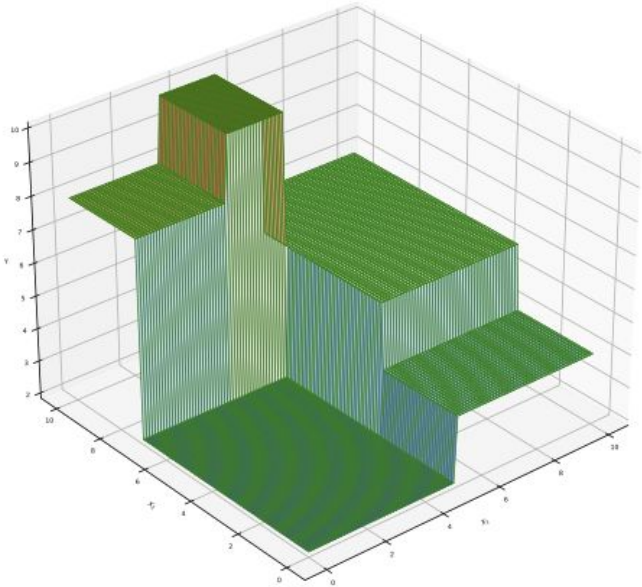
objects of class “no”

We can generalize to many features

https://probml.github.io/pml-book/book0.html

Decision Trees



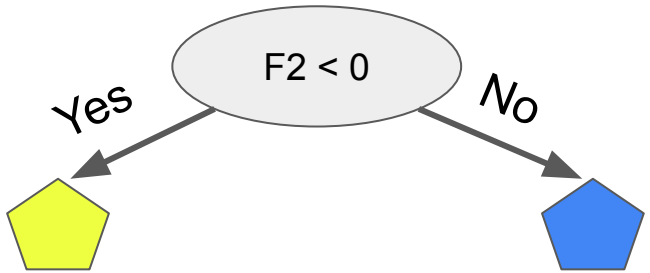


We can also apply decision trees to regression.

Decision Trees: Training

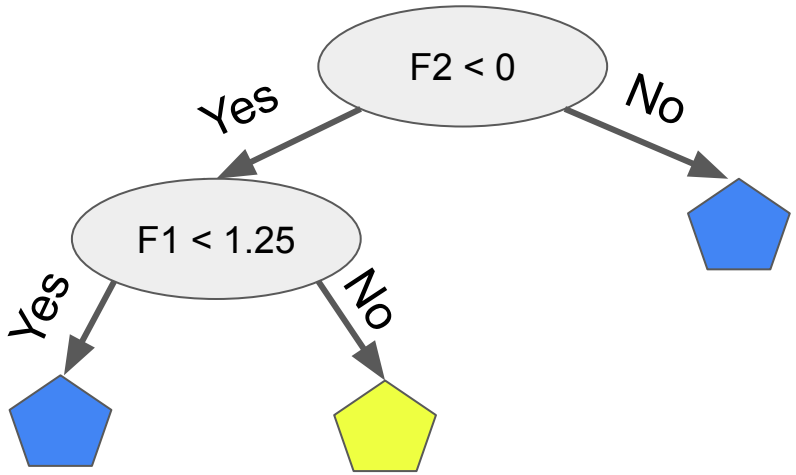
We will *grow* the decision tree: at each iteration, expand one of the leafs

Decision Trees: Training



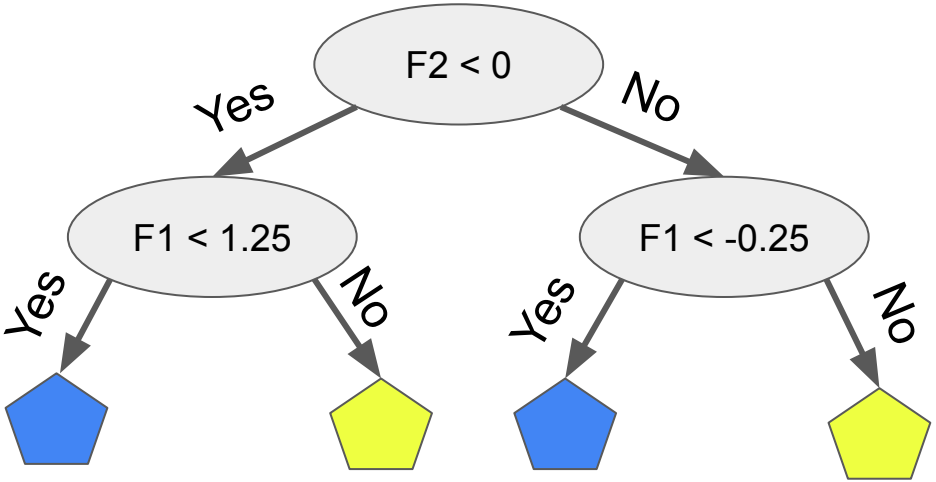
We will *grow* the decision tree: at each iteration, expand one of the leafs

Decision Trees: Training



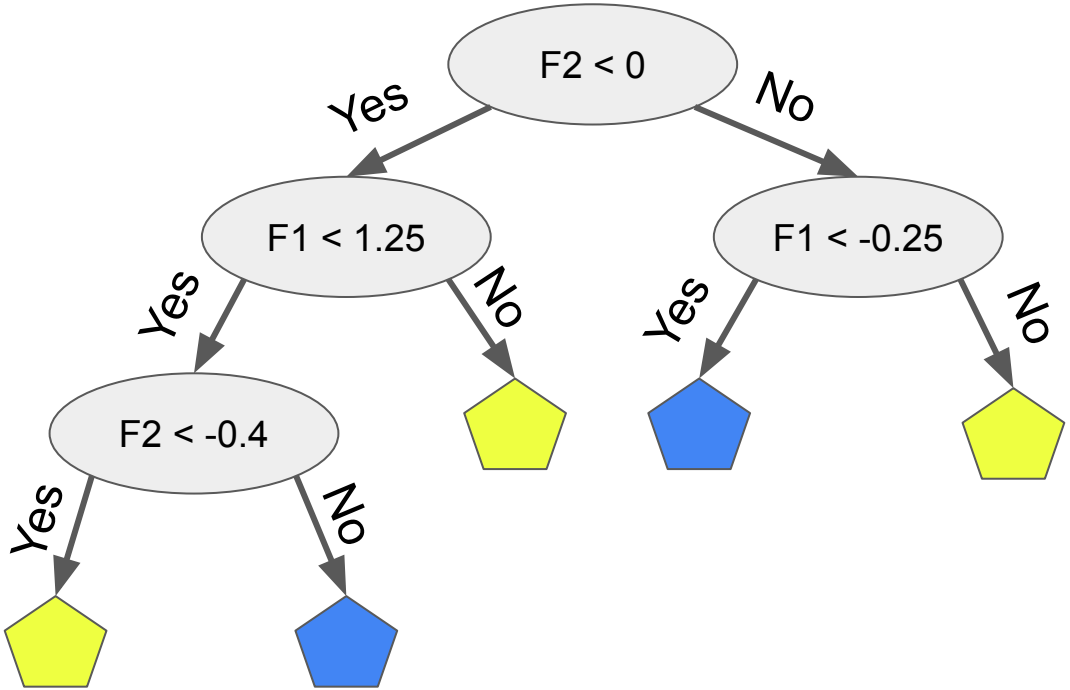
We will *grow* the decision tree: at each iteration, expand one of the leafs

Decision Trees: Training



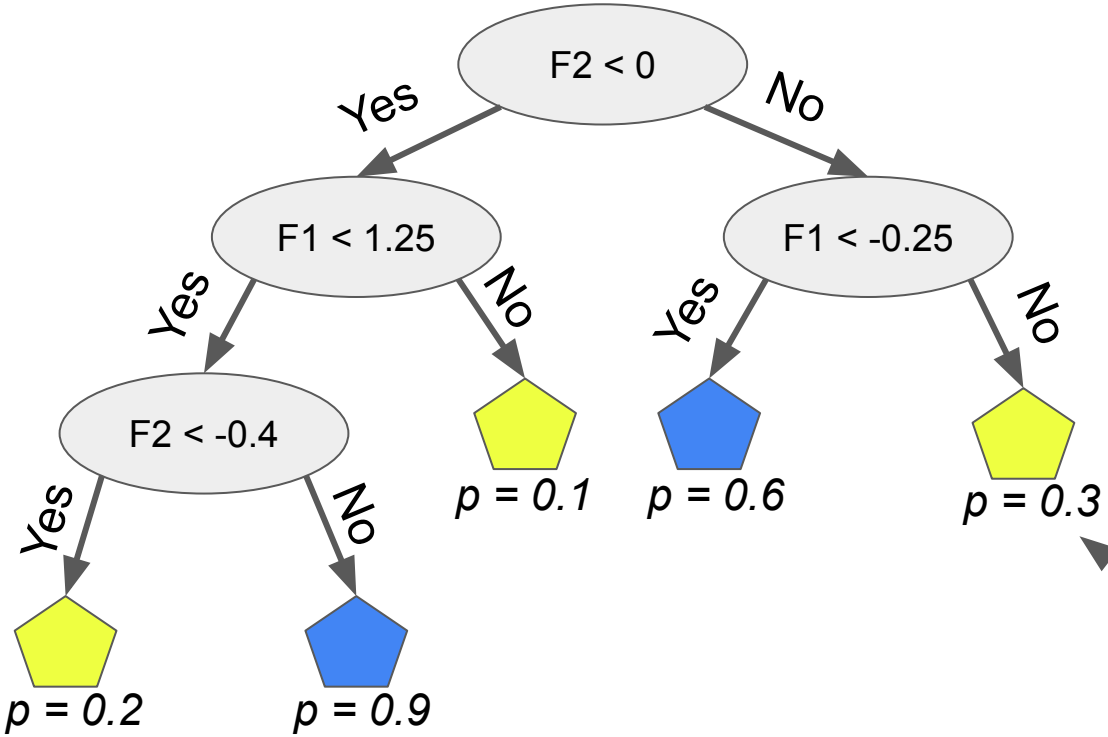
We will *grow* the decision tree: at each iteration, expand one of the leafs

Decision Trees: Training



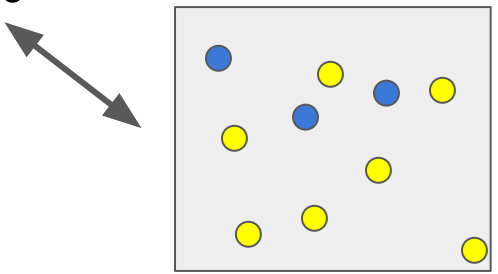
We will *grow* the decision tree: at each iteration, expand one of the leafs

Decision Trees: Training

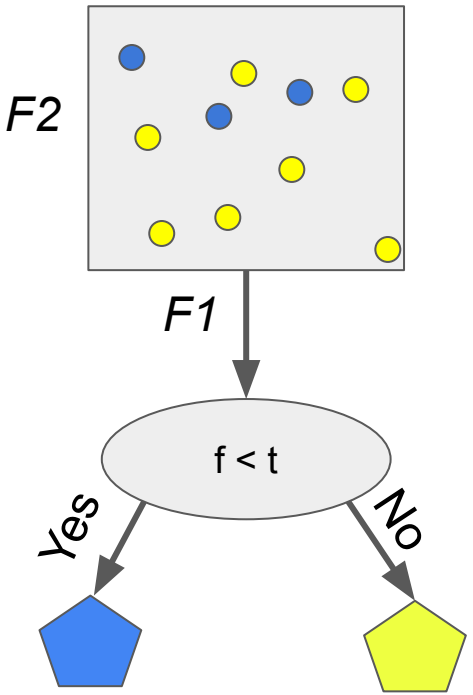


We will *grow* the decision tree: at each iteration, expand one of the leafs

One more detail: for classification the leaf nodes will have the fraction of class-1 datapoints.

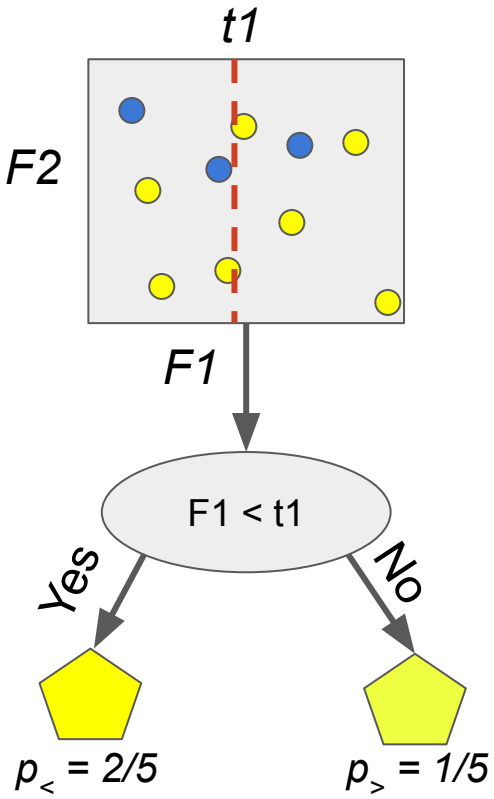


Decision Trees: Training



Expand a node: find (1) feature *f* and (2) threshold *t* so that we achieve the lowest *loss.*

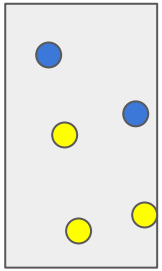
Decision Trees: Training



Expand a node: find (1) feature *f* and (2) threshold *t* so that we achieve the lowest *loss.*

$$L = \frac{N_<}{N} \cdot L(p_<) + \frac{N_>}{N} \cdot L(p_>)$$

Gini Impurity



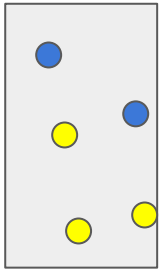
p = 2/5

We need to decide on *loss*, which will be applied per-region.

Gini impurity:

- sample a random object
- label it randomly,1 with probability *p* and 0 with *(1 - p)*
- what's the probability we will label it incorectly?

Gini Impurity



p = 2/5

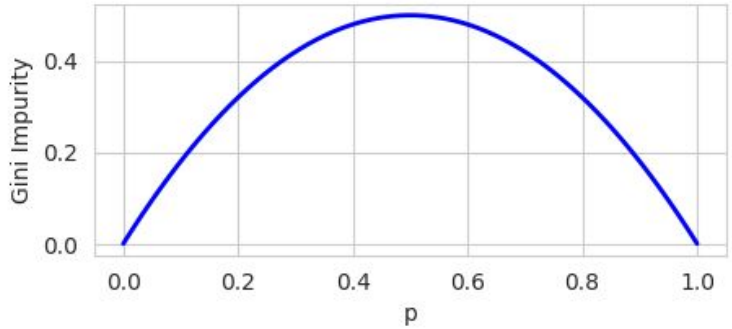
We need to decide on *loss*, which will be applied per-region.

Gini impurity:

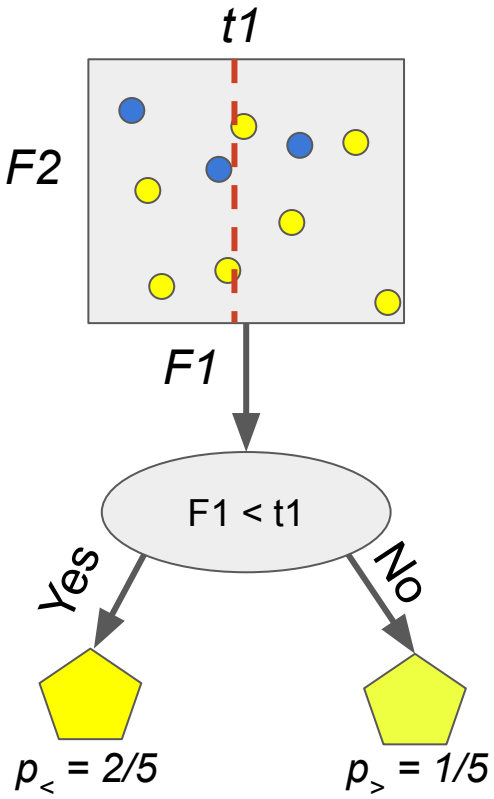
- sample a random object
- label it randomly,1 with probability *p* and 0 with *(1 - p)*
- what's the probability we will label it incorectly?

$$I_g(p) = 1 - p^2 - (1-p)^2 = 2(p - p^2)$$

Lowest loss when *p=0* or *p=1*



Decision Trees: Training

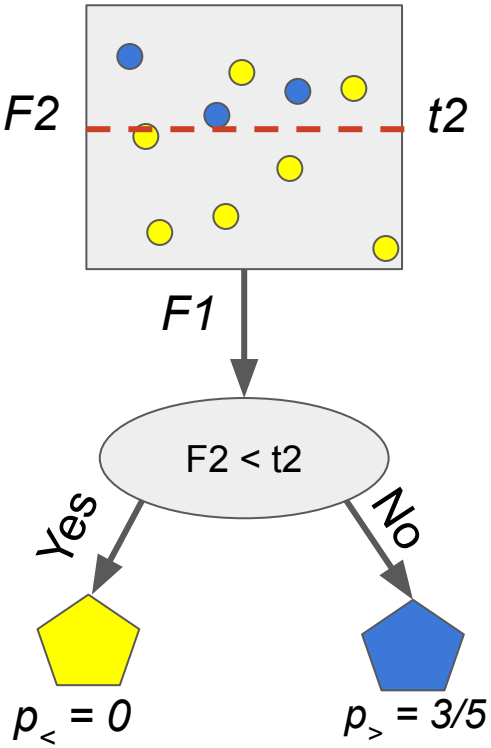


Expand a node: find (1) feature *f* and (2) threshold *t* so that we achieve the lowest *loss.*

$$L = \frac{2N_<}{N} \cdot (p_< - p_<^2) + \frac{2N_>}{N} \cdot (p_> - p_>^2)$$

$$L = 0.4$$

Decision Trees: Training



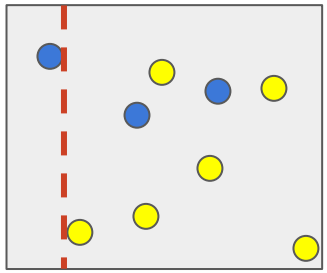
Expand a node: find (1) feature *f* and (2) threshold *t* so that we achieve the lowest *loss.*

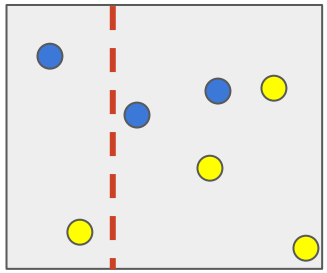
$$L = \frac{2N_<}{N} \cdot (p_< - p_<^2) + \frac{2N_>}{N} \cdot (p_> - p_>^2)$$

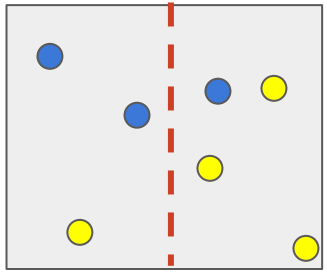
$$L = 0.24$$

We get a lower loss with F2, so we will use this feature.

Decision Trees: Training







For each feature, we need to try all thresholds.

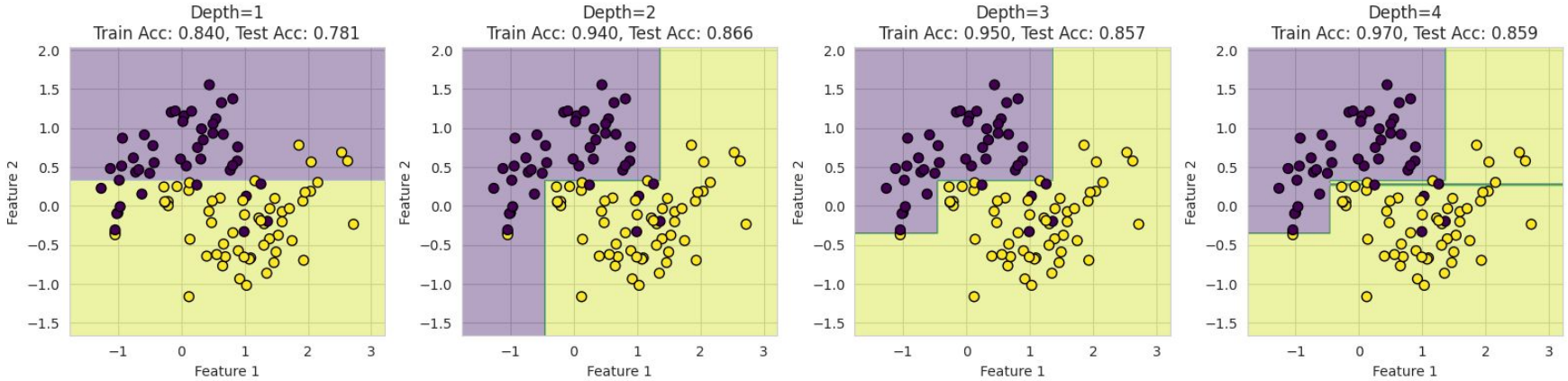
We try as many thresholds as datapoints in the node.

Decision Trees: Training

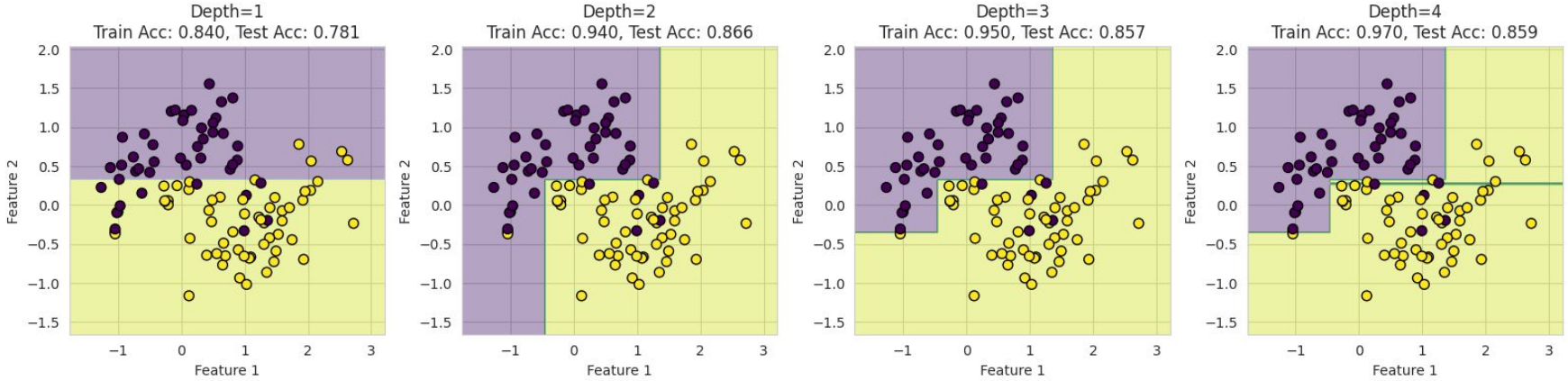
Tree growing algorithm

- For N iterations…
  - For each node…
    - For each feature *f*…
      - For each threshold *t*…
        - Compute the loss for the split for *(f, t)*
    - Select the best *(f, t)*, split the node into two correspondingly

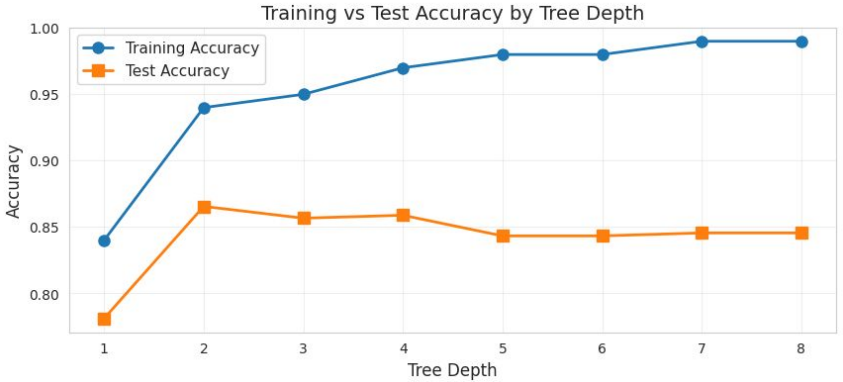
Decision Trees: Overfitting

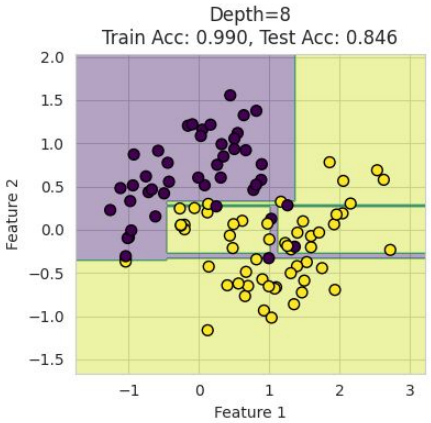


Decision Trees: Overfitting



We can overfit by making the tree too deep

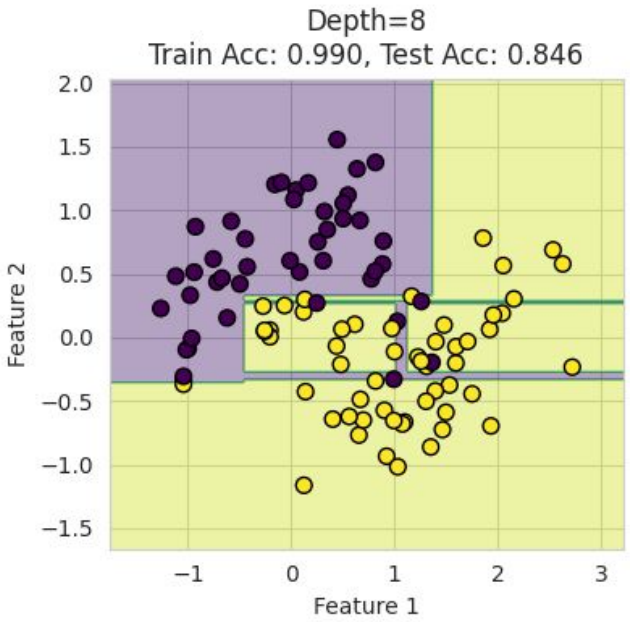




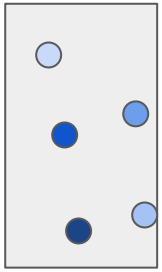
Decision Trees: Overfitting

To prevent overfitting:

- Limit tree depth
- Make sure each node has at least *K* datapoints
- Do not split a node if it has less than M datapoints
- Use random feature at each step instead of best
- …



Regression Criterion

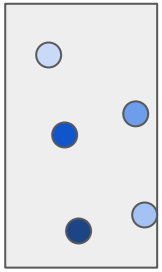


Prediction is the mean in the current region:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} y_i$$

What should our loss be?

Regression Criterion

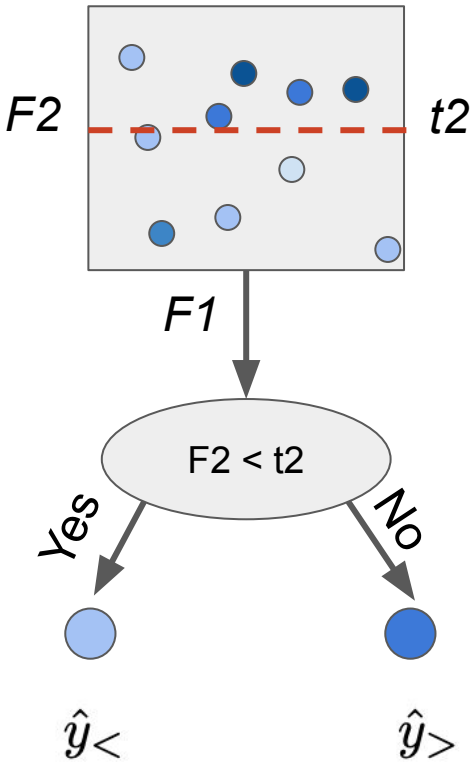


Prediction is the mean in the current region:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} y_i$$

What should our loss be? MSE

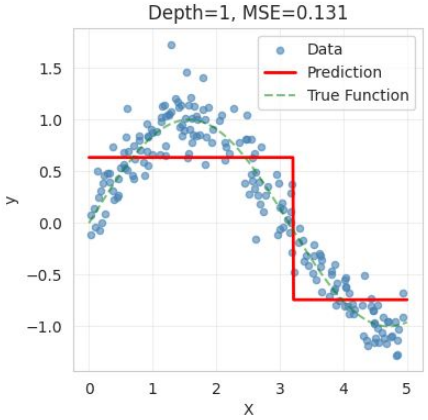
Decision Trees: Regression



Expand a node: find (1) feature *f* and (2) threshold *t* so that we achieve the lowest *loss.*

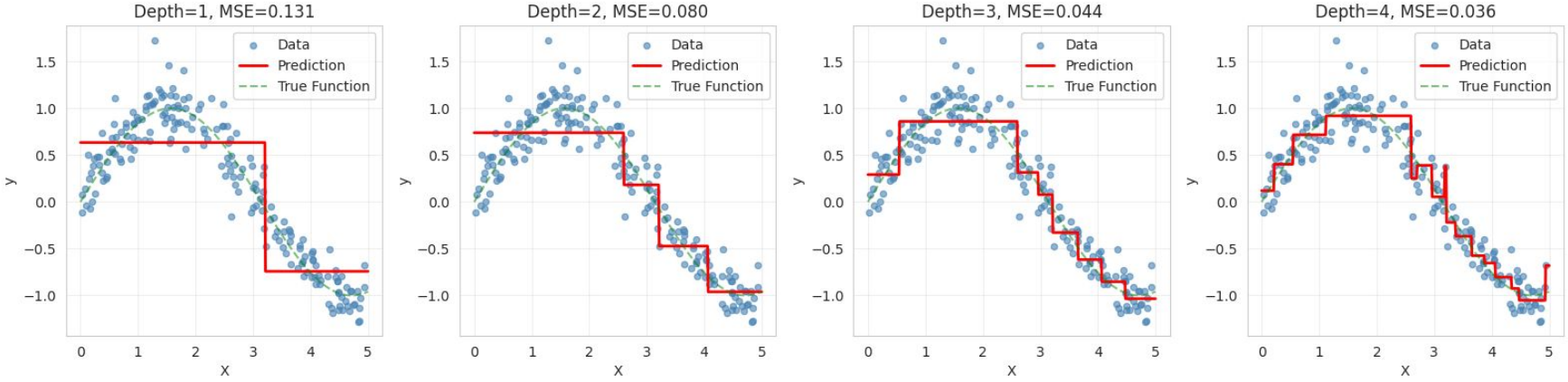
$$L = \frac{N_<}{N} \cdot \sum_{(x,y)\in\mathcal{D}_<} (y - \hat{y}_<)^2 + \frac{N_>}{N} \cdot \sum_{(x,y)\in\mathcal{D}_>} (y - \hat{y}_>)^2$$

Decision Trees: Regression



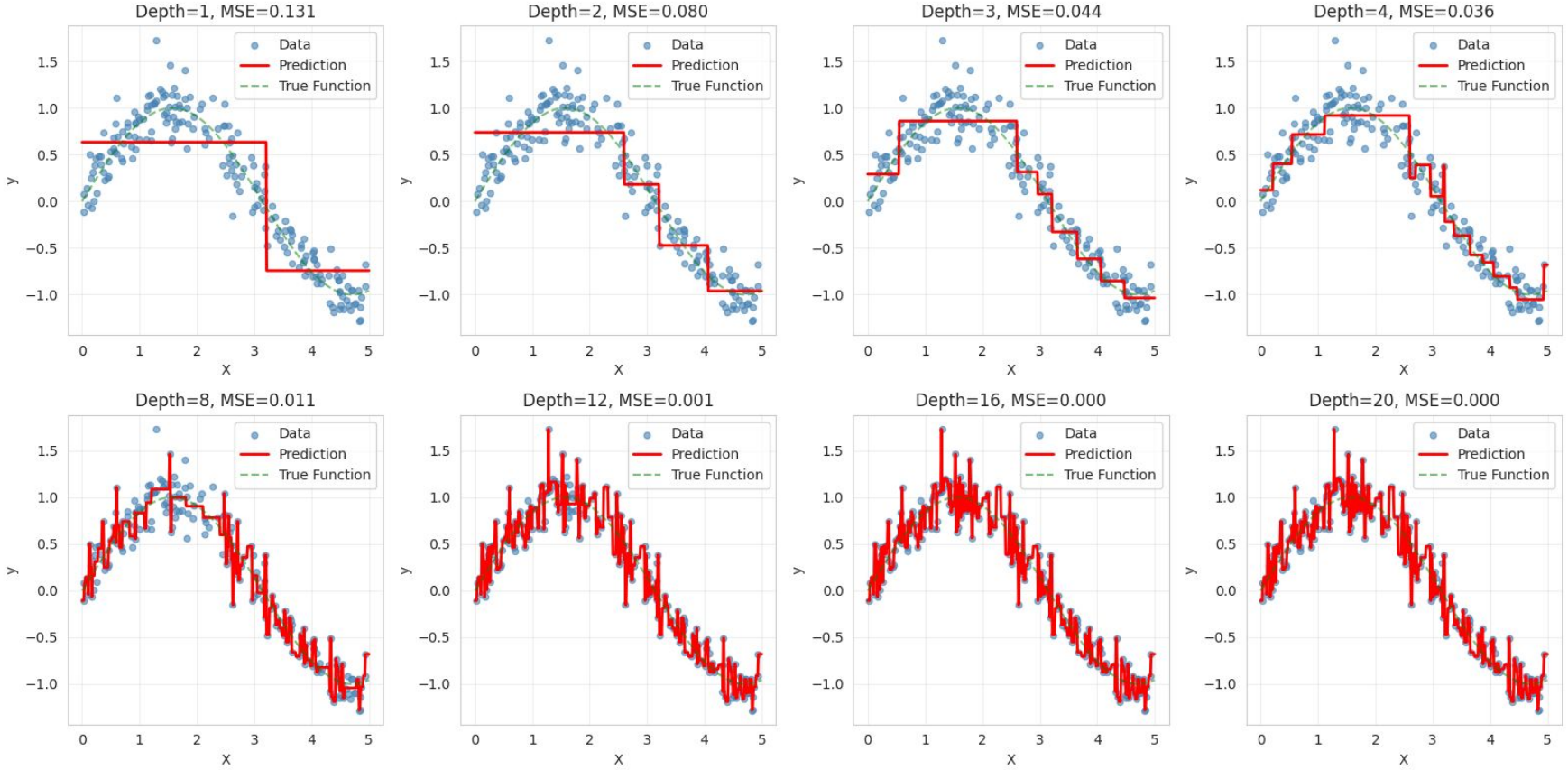
We get a piecewise constant fit. *For a tree of depth D, how many constant regions do we have?*

Decision Trees: Regression

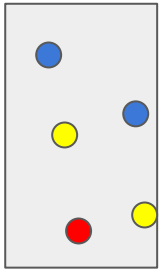


We get a piecewise constant fit. *For a tree of depth D, how many constant regions do we have?* 2^n

Decision Trees: Regression



Gini Impurity for Multi-Class



$p = 2/5$

$p = 2/5$

$p = 1/5$

We can generalize gini impurity to multi-class classification.

Gini impurity:

- sample a random object
- label it randomly,1 with probability *p* and 0 with *(1 - p)*
- what's the probability we will label it incorrectly?

$$I_g(p_1, \ldots, p_C) = \sum_{c=1}^{C} p_c \cdot (1 - p_c) = 1 - \sum_{c=1}^{C} p_c^2$$

Splitting Categorical Features

$N_{Cat} = 7$

$N_{Dog} = 2$

$N_{Platypus} = 17$

$N_{Parrot} = 4$

Decision trees can also work with categorical features

- Split: is_cat

Splitting Categorical Features

$N_{Cat} = 7$

$N_{Dog} = 2$

$N_{Platypus} = 17$

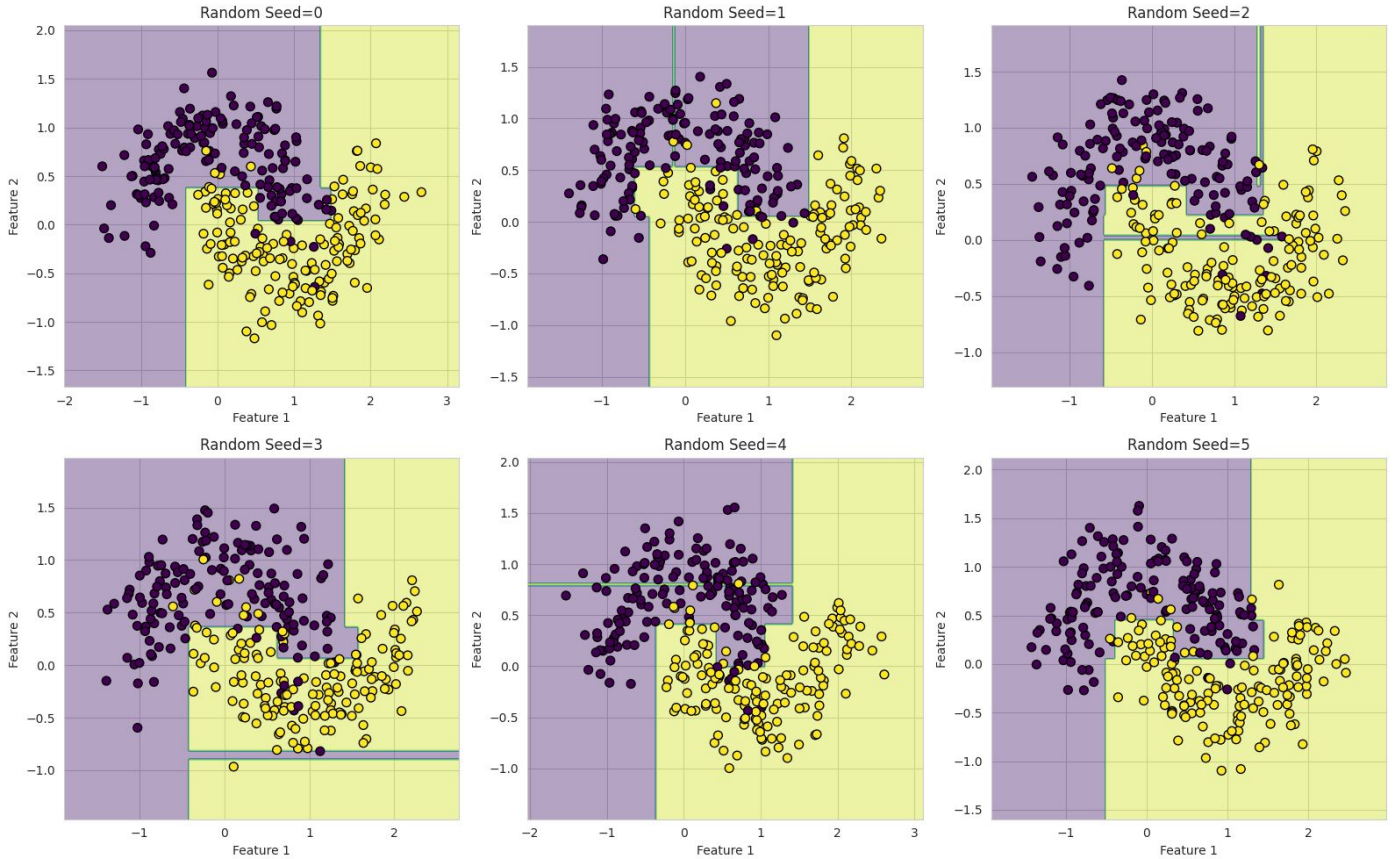
$N_{Parrot} = 4$

Decision trees can also work with categorical features

- Split: is_cat
- Split: is_cat or is_platypus

Decision Tree Instability

Small change in data ⇒ very different fit



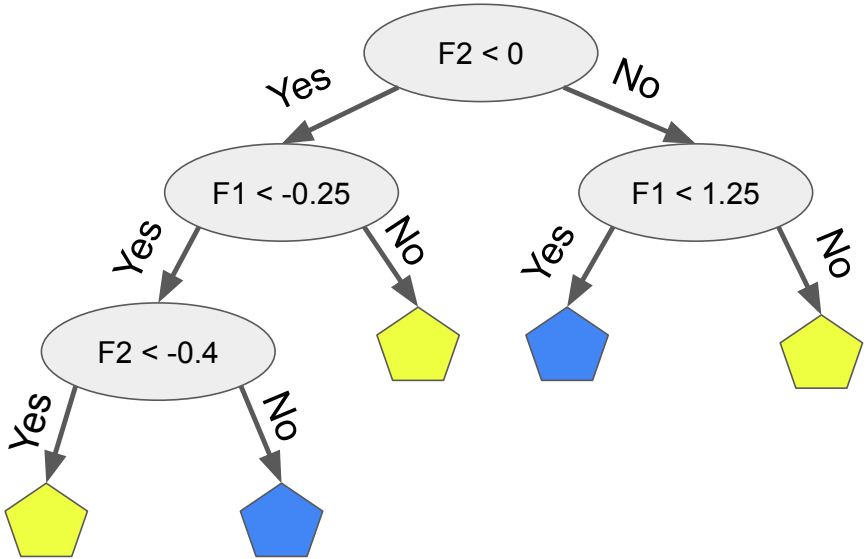
Decision Trees

Pros:

- Interpretable
- Can handle a mix of continuous and categorical features
- Very fast to fit
- Automatic variable selection

Cons:

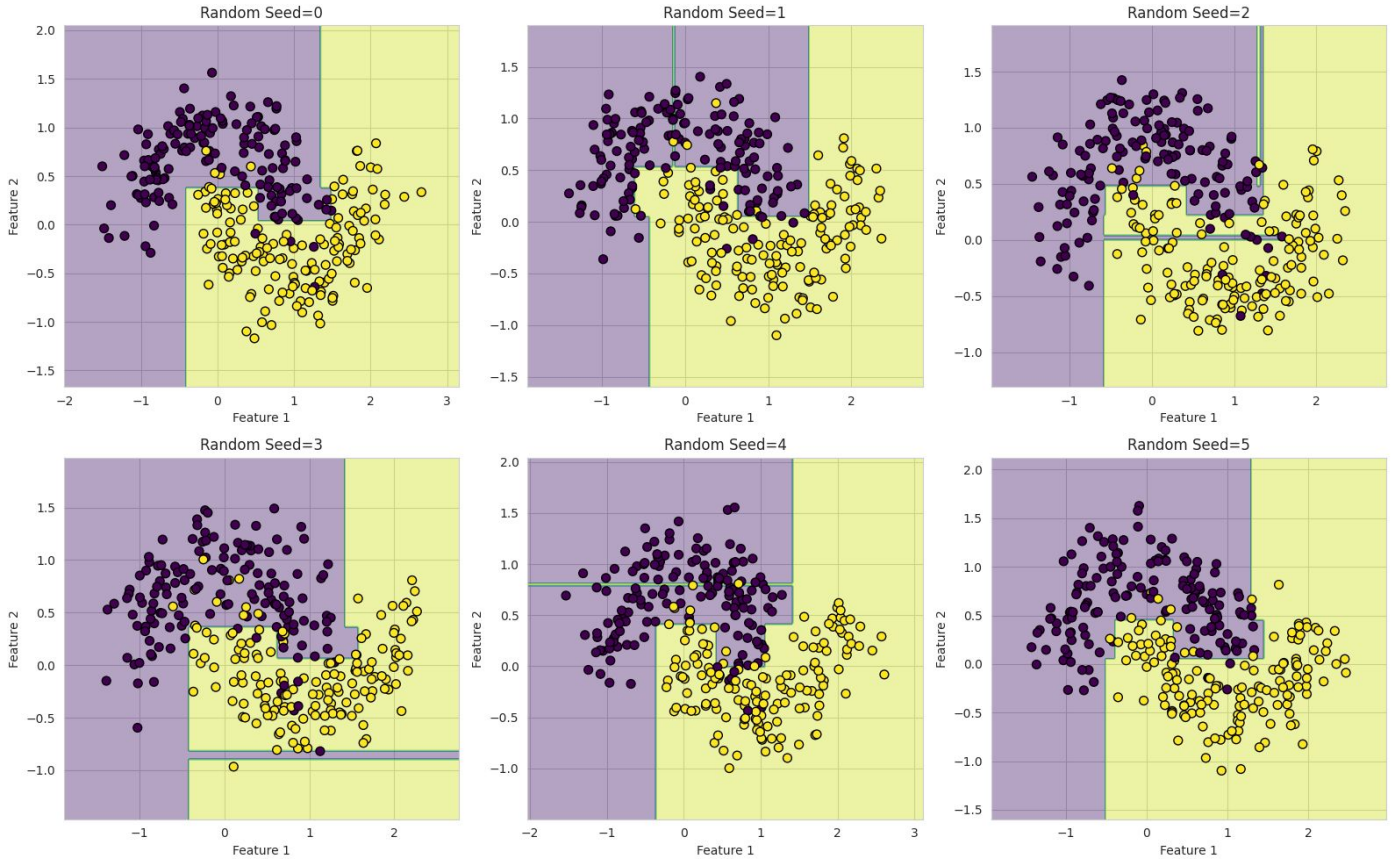
- Unstable
- Can overfit




Ensembling

Ensembling: Motivation

Idea 💡: train many trees on different subsets of data and average predictions



Ensembling: Motivation

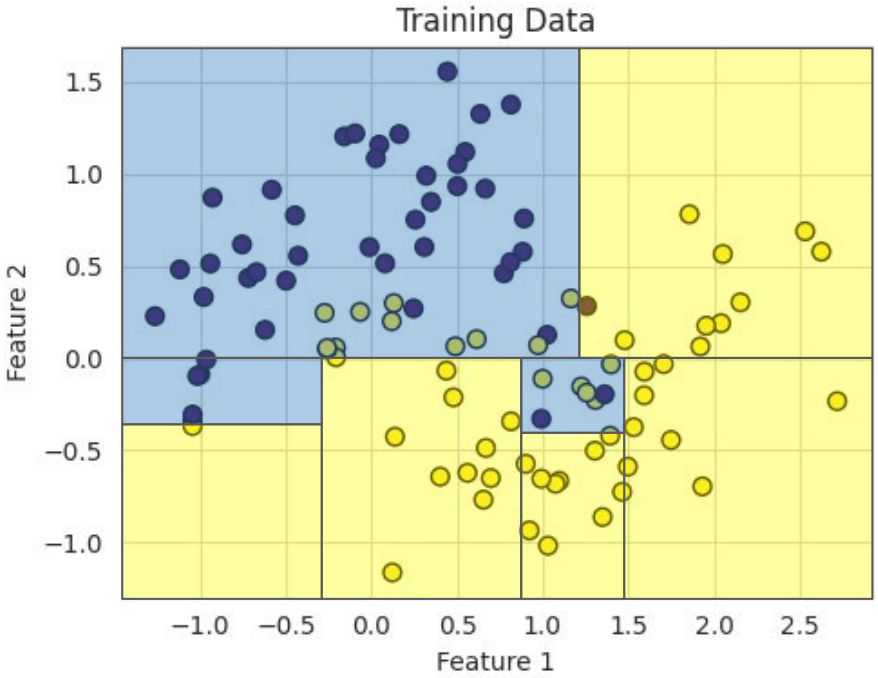
Ensemble: average predictions across multiple models

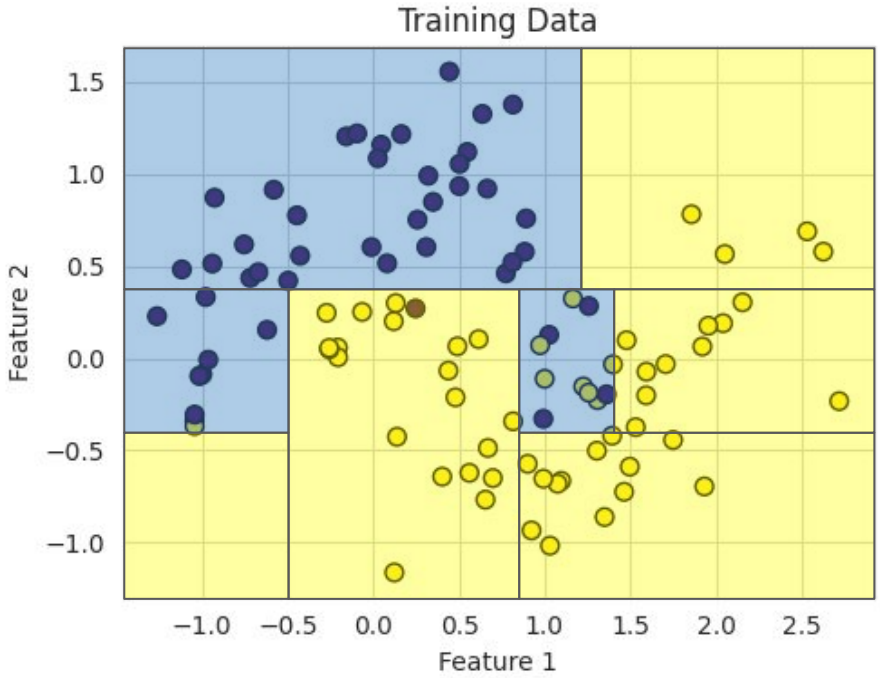
- For regression average predictions
- For classification average class probabilities

$$f(y|\boldsymbol{x}) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(y|\boldsymbol{x})$$

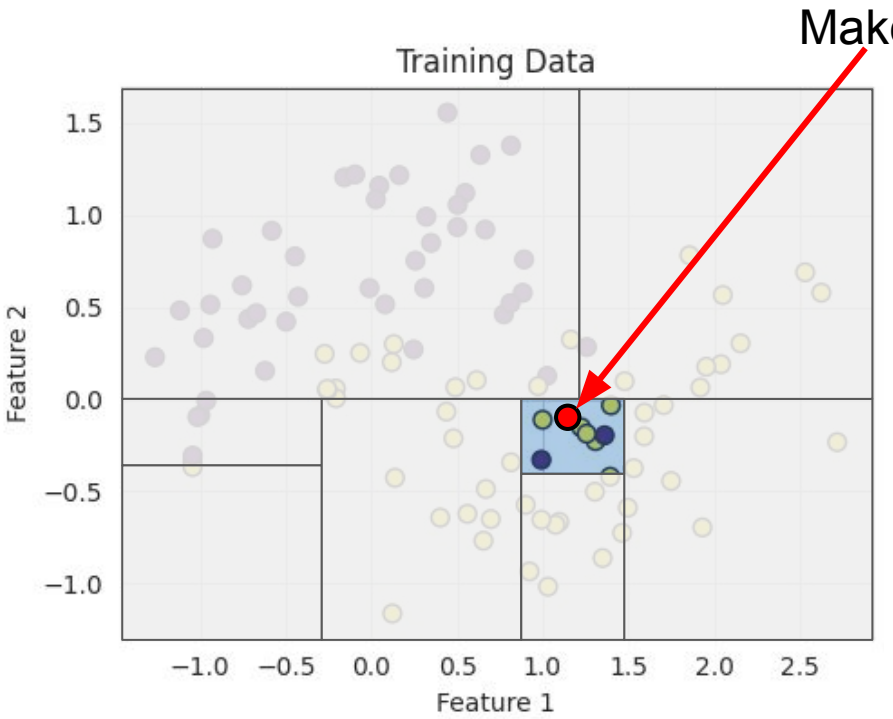
- Can do for any collection of models, generally leads to better performance

Bagging Algorithm

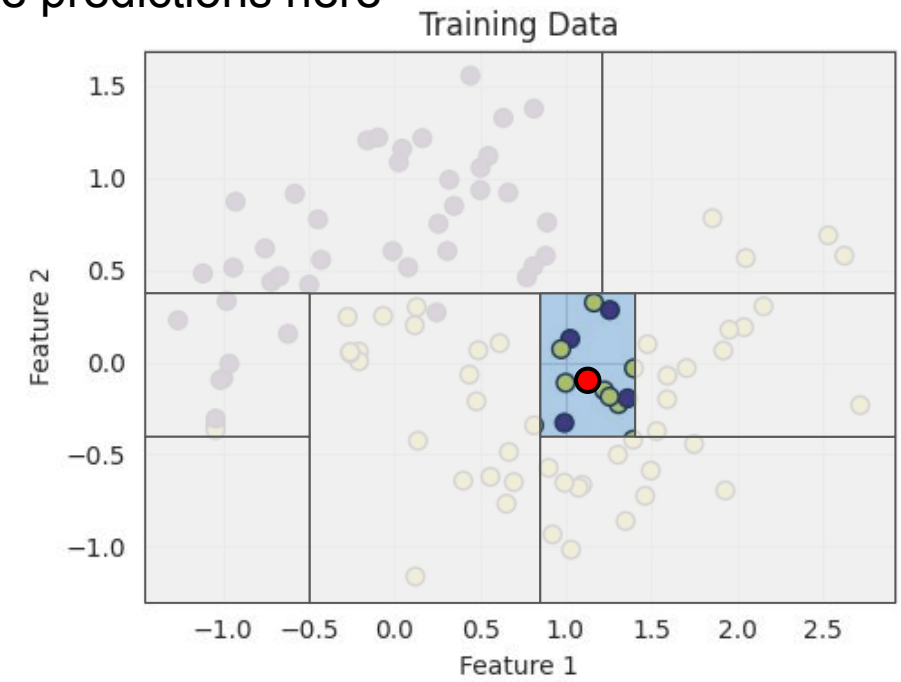




Bagging Algorithm



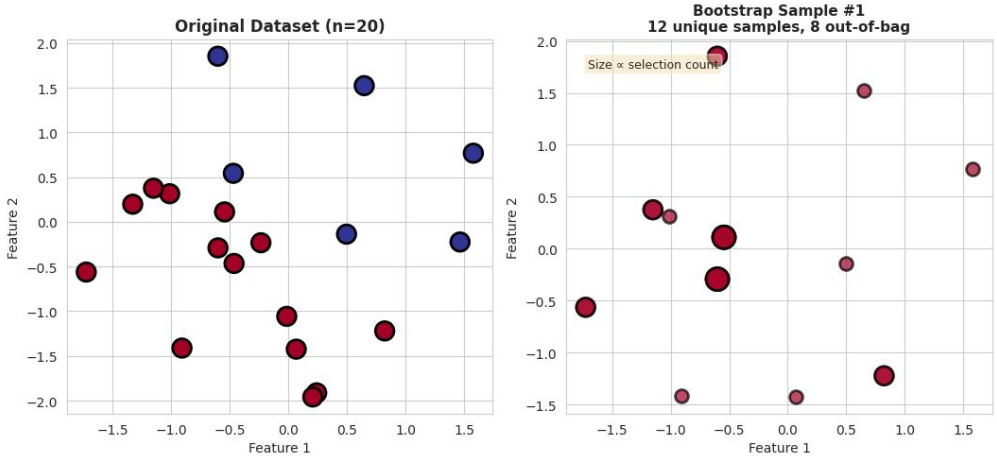
$p_1 = 0.29$



$p_2 = 0.36$

$p = (0.29 + 0.36) / 2 = 0.32$

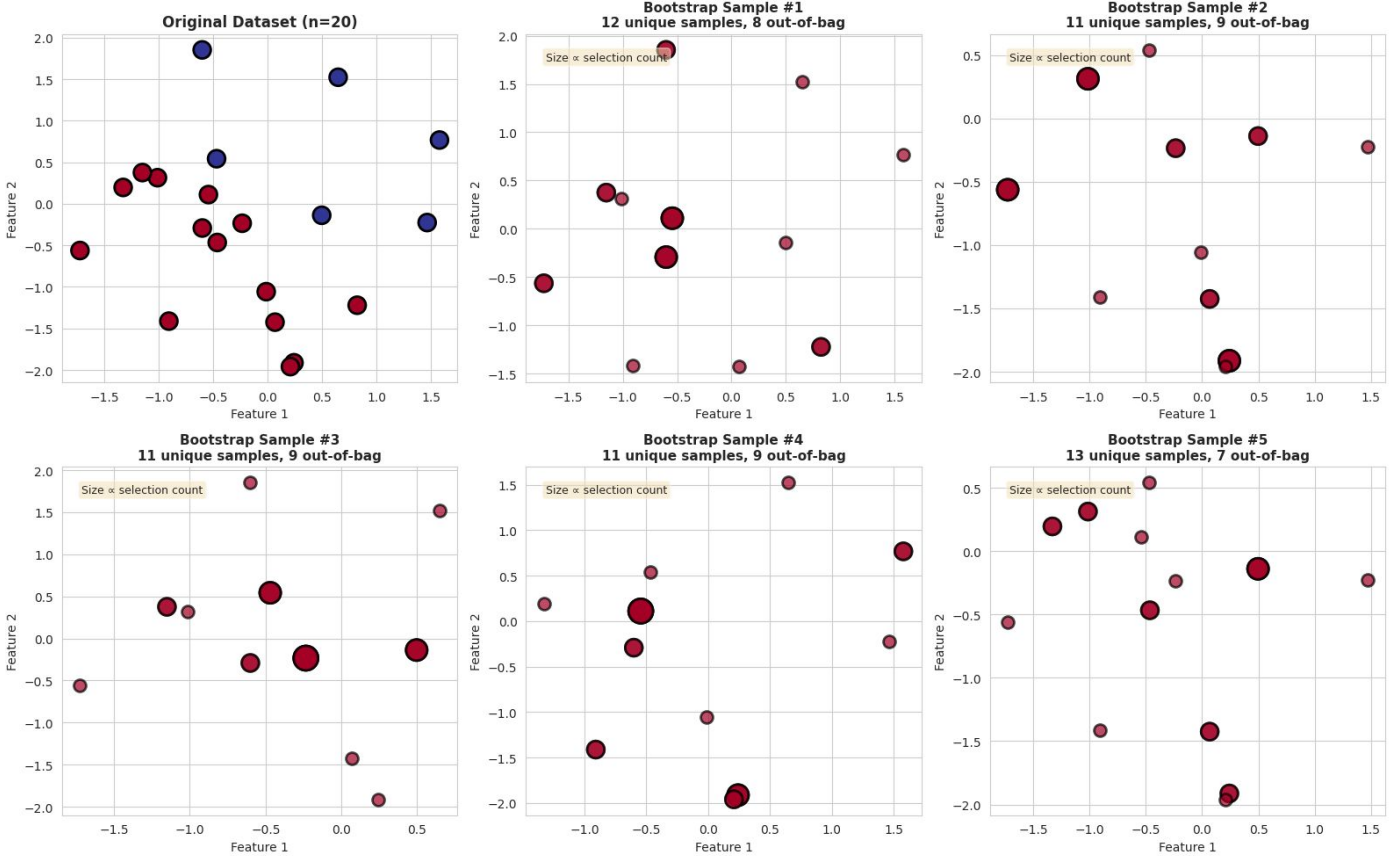
Bagging: Bootstrap Aggregating



We will sample *N* datapoints from our dataset *with replacement*

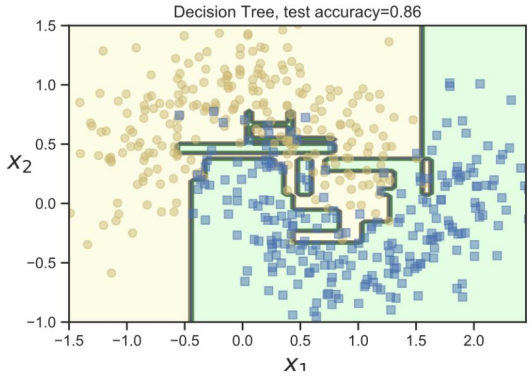
- Some datapoints will not appear
- Some datapoints will appear multiple times

Bagging: Bootstrap Aggregating

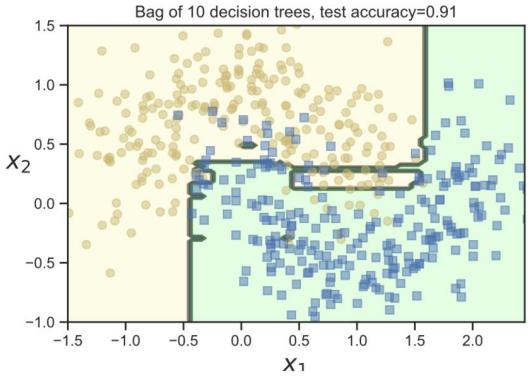


Bagging: Bootstrap Aggregating

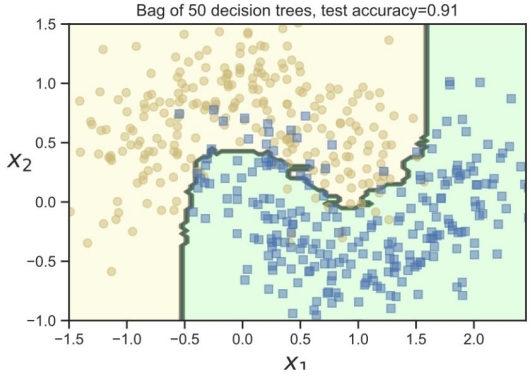
- Bagging finds smoother decision boundaries
- More stable
- Higher accuracy



(a)



(b)



Bagging Algorithm

Bagging:

- Repeat *B* times: bootstrap a random dataset, train tree i
  - For N iterations…
    - For each node n…
      - For each feature *f*…
        - For each threshold *t*…
          - Compute the loss for the split for *(f, t)*
      - Select the best *(f, t)*, split the node into two correspondingly
- For a test input, compute predictions of all trees; average them to get final output

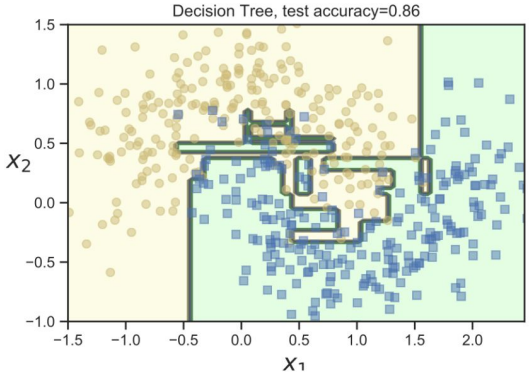
Random Forest

Bagging:

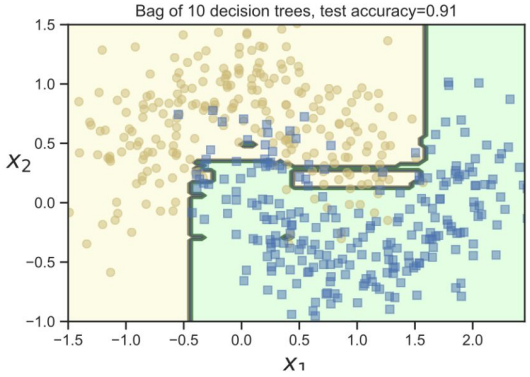
- Repeat *B* times: bootstrap a random dataset, train tree i
  - For N iterations…
    - For each node n…
      - For each feature *f* in a random subset $F_{n,i}$…
        - For each threshold *t*…
          - Compute the loss for the split for *(f, t)*
      - Select the best *(f, t)*, split the node into two correspondingly
- For a test input, compute predictions of all trees; average them to get final output

Extra randomness by using random feature subsets

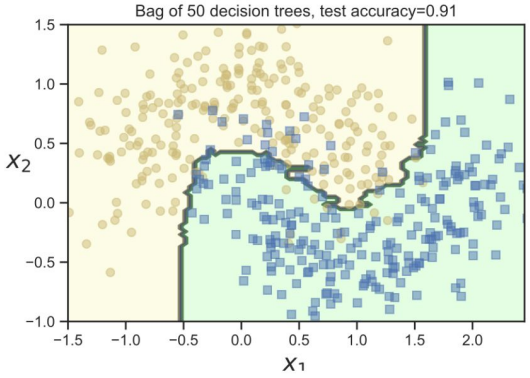
Bagging and Random Forests

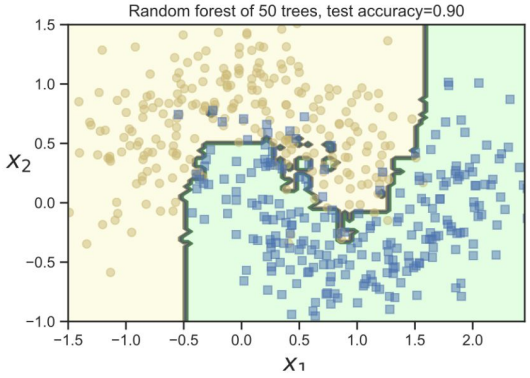


(a)



(b)





Bagging and Random Forests

Bagging and random forests are both examples of *parallel* ensembling

- Train multiple models with some source of randomness
  - Data (bootstrapping)
  - Random features
  - Randomness in the training algorithm (random seed)

- At test time average predictions over all models in the ensemble

$$f(y|\boldsymbol{x}) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(y|\boldsymbol{x})$$

Ensembling is a powerful and universal idea!

Bagging and Random Forests

Random Forests hyperparameters

- Number of trees (more is better, ~100-500)
- Max depth (~10-30)
- Max features (how many features we consider per split)
- …

Bagging and Random Forests

Random Forests:

- Very robust to overfitting
- Easy to use, fast
- Can handle high-dimensional mixed continuous-discrete data
- Less interpretable than decision trees


Boosting

Boosting

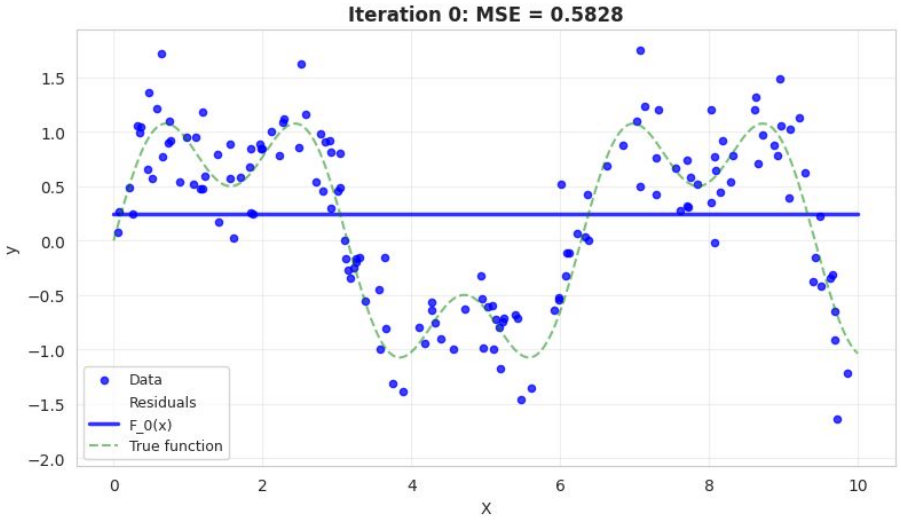
We will fit an additive model: $f(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{m=1}^{M} \beta_m F_m(\boldsymbol{x};\boldsymbol{\theta}_m)$

We will minimize the loss: $\mathcal{L}(f) = \sum_{i=1}^{N} \ell(y_i, f(\boldsymbol{x}_i))$

Boosting: fit ensemble members F_i sequentially, depending on each other

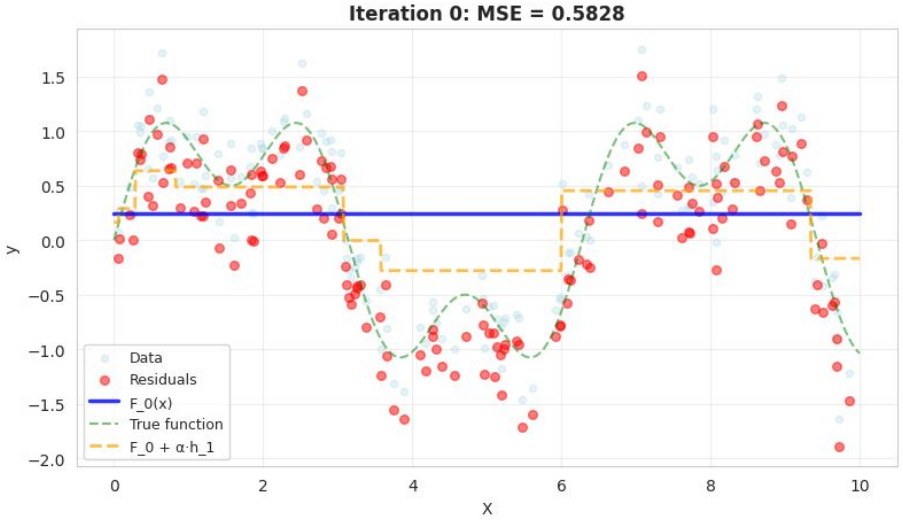
- Fit F_1 on original data
- Train F_2 to fit the *residual errors* of F_1

Gradient Boosting



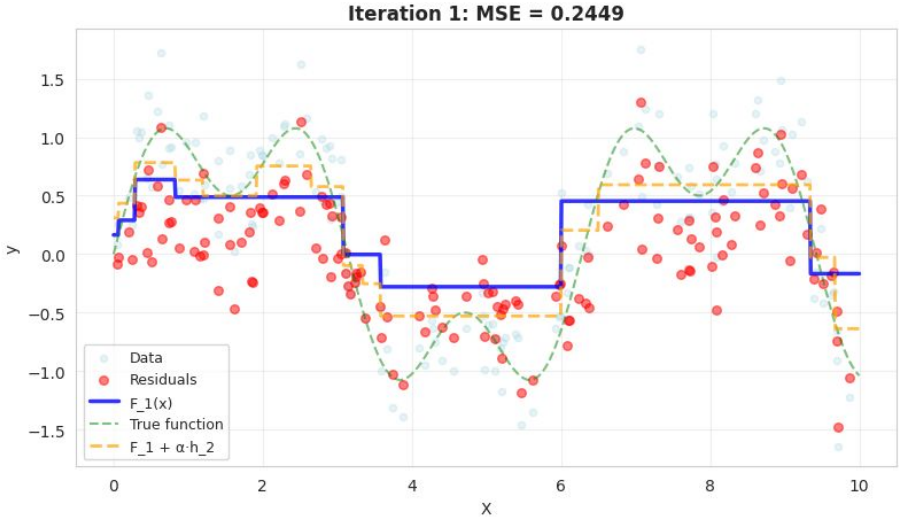
- Fit a tree $F_1(x)$ *to the data*

Gradient Boosting



- Fit a tree $F_1(x)$ *to the data*
- Compute the residuals $y_i - F_1(x_i)$; fit tree h_1 to the residuals
- $F_2(x) = F_1(x) + \alpha\, h_1(x)$

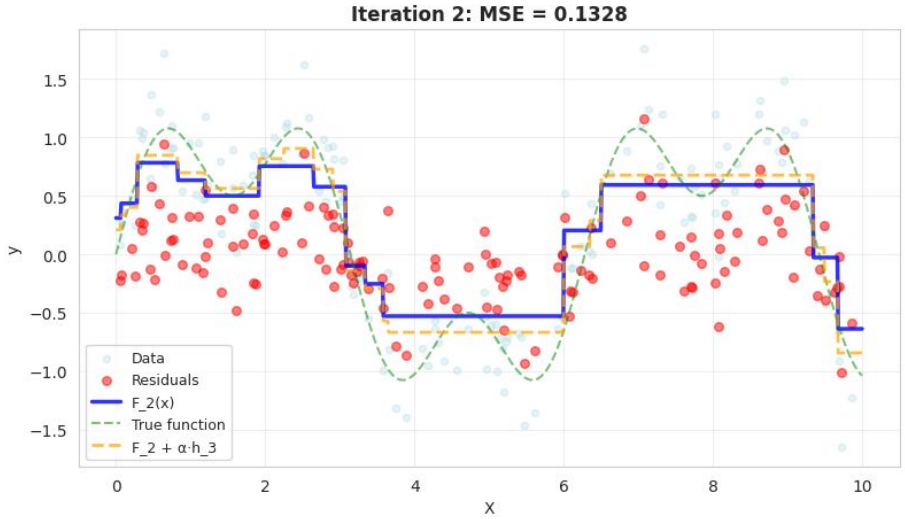
Gradient Boosting



- Fit a tree $F_1(x)$ *to the data*
- Compute the residuals $y_i - F_1(x_i)$; fit tree h_1 to the residuals
- $F_2(x) = F_1(x) + \alpha\, h_1(x)$

  …
- Compute the residuals $y_i - F_m(x_i)$; fit tree h_m to the residuals
- $F_{m+1}(x) = F_m(x) + \alpha\, h_m(x)$

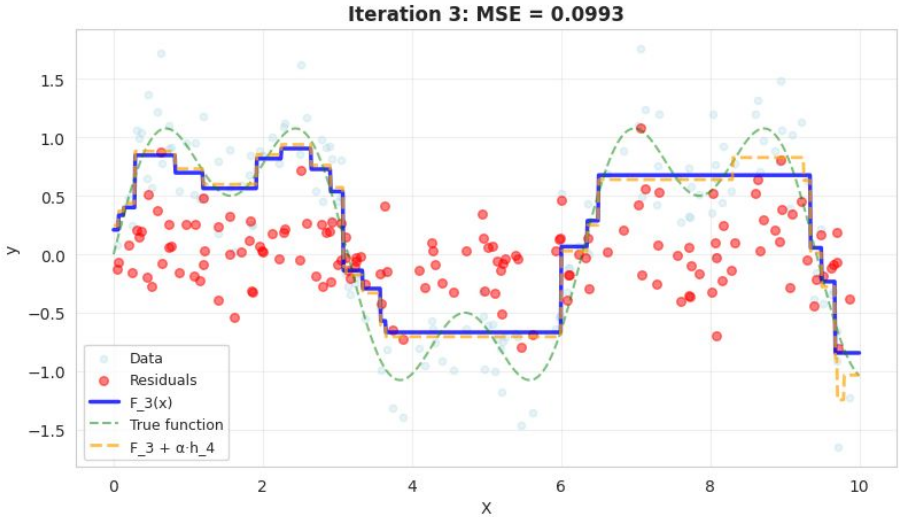
Gradient Boosting



- Fit a tree $F_1(x)$ *to the data*
- Compute the residuals $y_i - F_1(x_i)$; fit tree h_1 to the residuals
- $F_2(x) = F_1(x) + \alpha\, h_1(x)$

  …
- Compute the residuals $y_i - F_m(x_i)$; fit tree h_m to the residuals
- $F_{m+1}(x) = F_m(x) + \alpha\, h_m(x)$

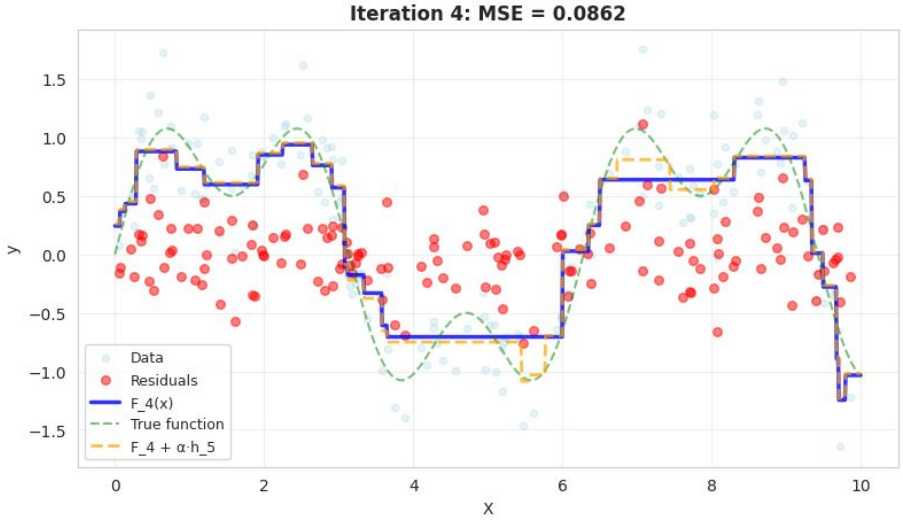
Gradient Boosting



- Fit a tree $F_1(x)$ *to the data*
- Compute the residuals $y_i - F_1(x_i)$; fit tree h_1 to the residuals
- $F_2(x) = F_1(x) + \alpha\, h_1(x)$

  …
- Compute the residuals $y_i - F_m(x_i)$; fit tree h_m to the residuals
- $F_{m+1}(x) = F_m(x) + \alpha\, h_m(x)$

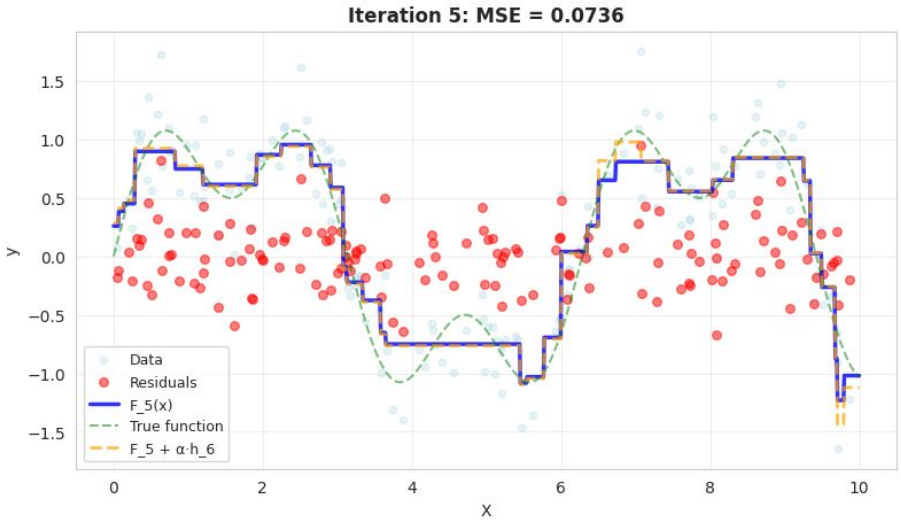
Gradient Boosting



- Fit a tree $F_1(x)$ *to the data*
- Compute the residuals $y_i - F_1(x_i)$; fit tree h_1 to the residuals
- $F_2(x) = F_1(x) + \alpha\, h_1(x)$

  …
- Compute the residuals $y_i - F_m(x_i)$; fit tree h_m to the residuals
- $F_{m+1}(x) = F_m(x) + \alpha\, h_m(x)$

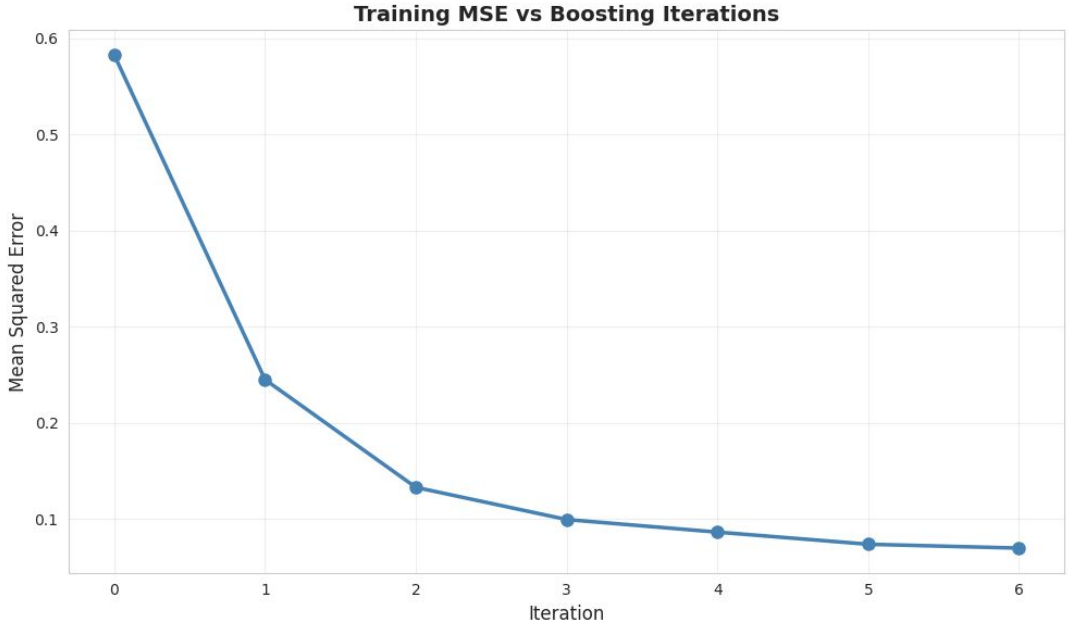
Gradient Boosting



- Fit a tree $F_1(x)$ *to the data*
- Compute the residuals $y_i - F_1(x_i)$; fit tree h_1 to the residuals
- $F_2(x) = F_1(x) + \alpha\, h_1(x)$

  …
- Compute the residuals $y_i - F_m(x_i)$; fit tree h_m to the residuals
- $F_{m+1}(x) = F_m(x) + \alpha\, h_m(x)$

Gradient Boosting



At each iteration, we are reducing the train loss by fitting the residuals.

Gradient Boosting

$$L(y_i, F(x_i)) = (y_i - F(x_i))^2$$

$$\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

More generally, we want to do gradient descent in the space of predictions on the training data!

The residual errors are gradients for the MSE loss

$$F_{m+1}(x_i) = F_{m+1}(x_i) + \alpha h_m(x_i) \approx F_{m+1}(x_i) + \alpha \left. \frac{\partial L(y_i, f)}{\partial f} \right|_{f=F_m(x_i)}$$

Gradient Boosting

$$L(y_i, F(x_i)) = (y_i - F(x_i))^2$$

$$\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

More generally, we want to do gradient descent in the space of predictions on the training data!

The residual errors are gradients for the MSE loss

$$F_{m+1}(x_i) = F_{m+1}(x_i) + \alpha h_m(x_i) \approx F_{m+1}(x_i) + \alpha \left. \frac{\partial L(y_i, f)}{\partial f} \right|_{f=F_m(x_i)}$$

Approximate gradient descent step!

Approximate, because h_m will not fit the prediction gradient exactly.

Gradient Boosting

Algorithm 18.3: Gradient boosting

1 Initialize $f_0(\boldsymbol{x}) = \text{argmin}_F \sum_{i=1}^N L(y_i, F(\boldsymbol{x}_i))$

2 **for** $m = 1 : M$ **do**

3 Compute the gradient residual using $r_{im} = -\left[\frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f(\boldsymbol{x}_i)=f_{m-1}(\boldsymbol{x}_i)}$

4 Use the weak learner to compute $F_m = \text{argmin}_F \sum_{i=1}^N (r_{im} - F(\boldsymbol{x}_i))^2$

5 Update $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \nu F_m(\boldsymbol{x})$

6 Return $f(\boldsymbol{x}) = f_M(\boldsymbol{x})$

Gradient Boosting: General Loss

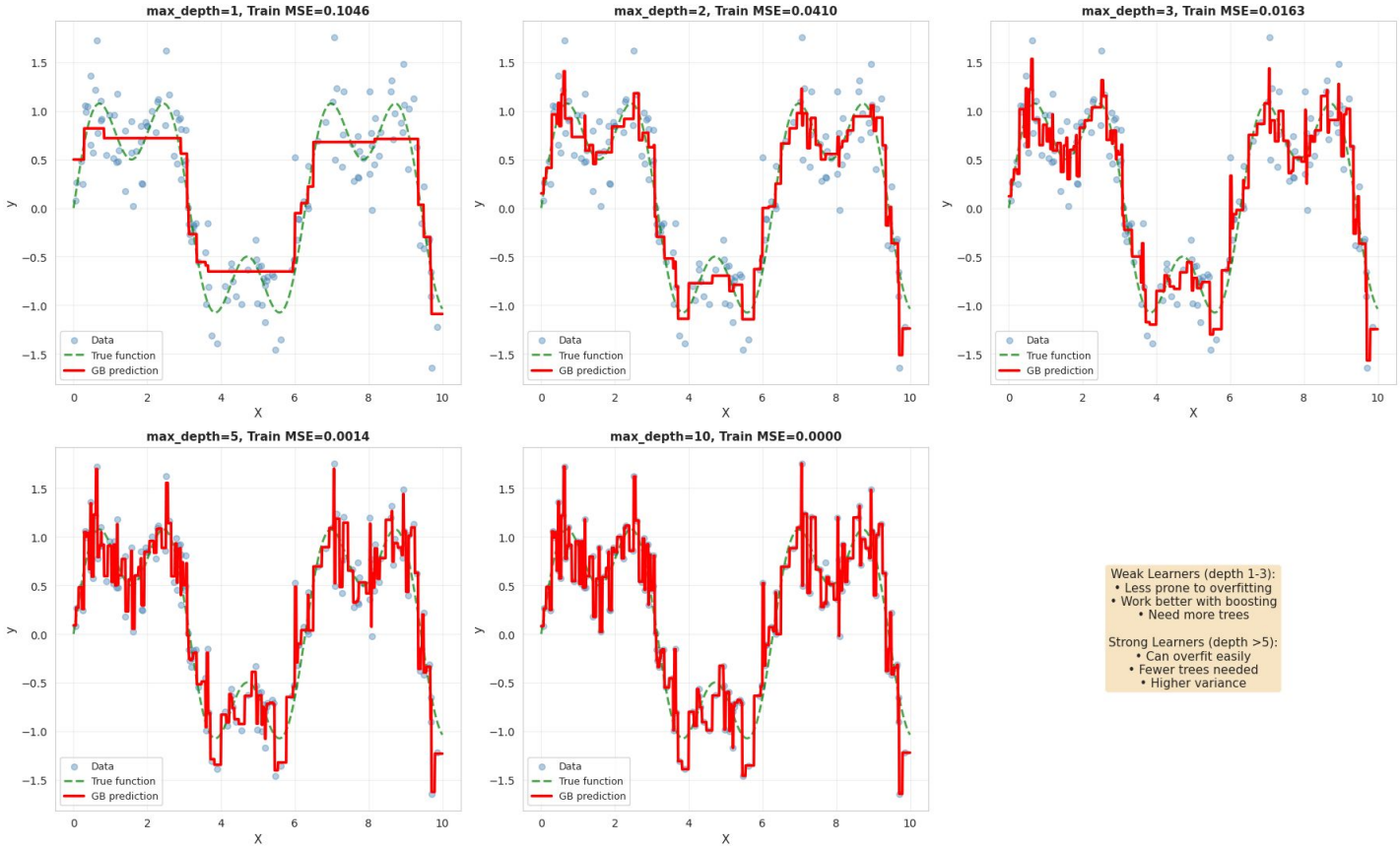
- Fit a tree $F_1(x)$ *to the data*
- For N iterations…
  - Compute the gradients $\left.\frac{\partial L(y_i, f)}{\partial f}\right|_{f=F_m(x_i)}$
  - Fit tree h_m to the gradients
  - $F_{m+1}(x_i) = F_{m+1}(x_i) + \alpha h_m(x_i)$

Gradient Boosting

Name	Loss	$-\partial \ell(y_i, f(\boldsymbol{x}_i))/\partial f(\boldsymbol{x}_i)$
Squared error	$\frac{1}{2}(y_i - f(\boldsymbol{x}_i))^2$	$y_i - f(\boldsymbol{x}_i)$
Absolute error	$\lvert y_i - f(\boldsymbol{x}_i)\rvert$	$\operatorname{sgn}(y_i - f(\boldsymbol{x}_i))$
Exponential loss	$\exp(-\tilde{y}_i f(\boldsymbol{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\boldsymbol{x}_i))$
Binary Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$
Multiclass logloss	$-\sum_c y_{ic} \log \pi_{ic}$	$y_{ic} - \pi_{ic}$

We can apply gradient boosting with any loss function and any underlying model class (weak learner). Typically, people use decision trees.

Gradient Boosting: Practical Considerations

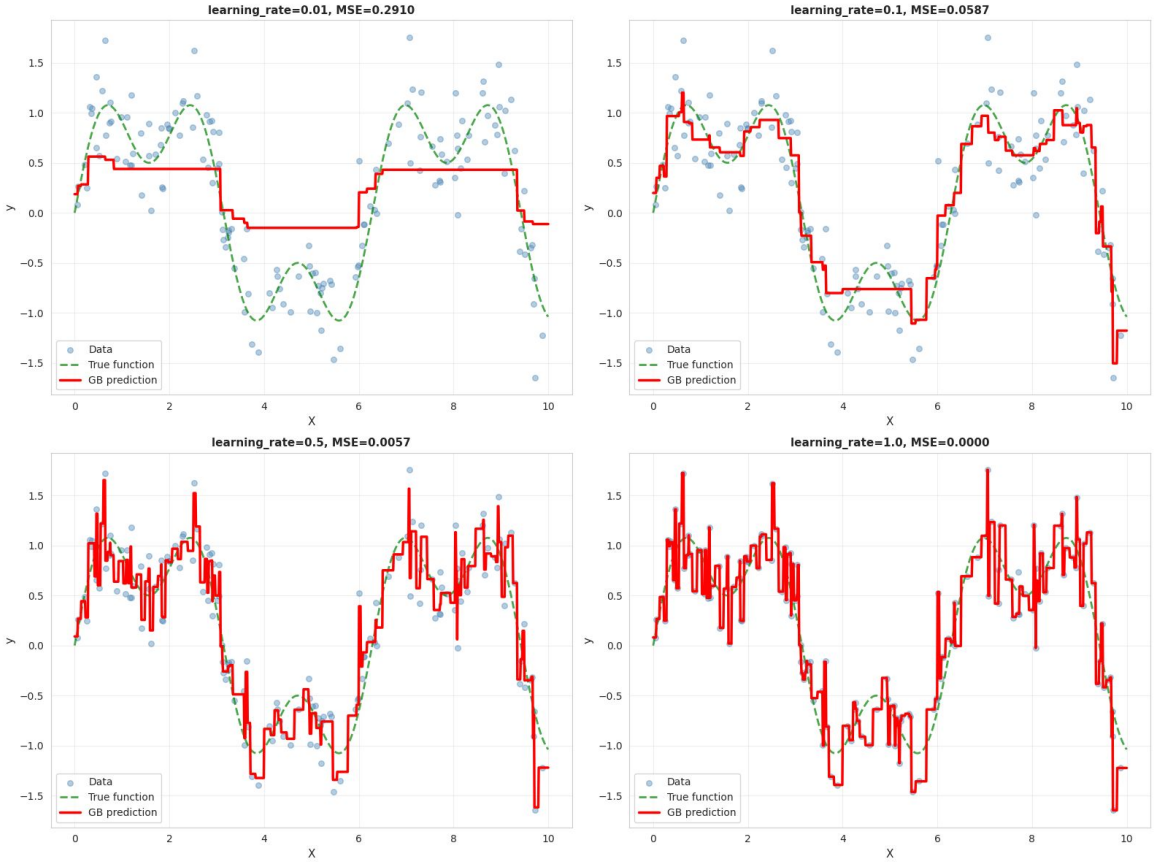


Weak Learners (depth 1-3):
- Less prone to overfitting
- Work better with boosting
- Need more trees

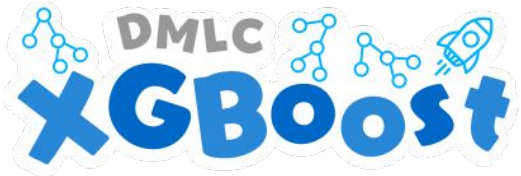
Strong Learners (depth >5):
- Can overfit easily
- Fewer trees needed
- Higher variance

Gradient Boosting: Practical Considerations

- Generally, we need to tune the learning rate
- Too high ⇒ updates too big
- Too low ⇒ updates too small



Gradient Boosting

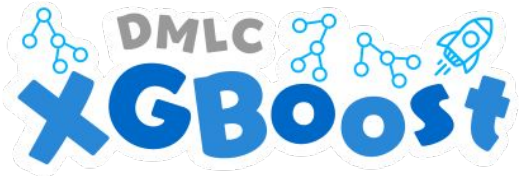


Yandex
CatBoost

Methods based on gradient boosting of trees are very practical. That's usually the first thing you should try on *tabular* tasks.

https://github.com/dmlc/xgboost
https://catboost.ai/

Gradient Boosting



Yandex
CatBoost

CatBoost is an algorithm for **gradient boosting on decision trees**. It is developed by Yandex researchers and engineers, and is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in other companies, including CERN, Cloudflare, Careem taxi. It is in open-source and can be used by anyone.

https://github.com/dmlc/xgboost
https://catboost.ai/

Gradient Boosting

Gradient Boosting hyperparameters

- Number of trees (more is better, ~100-500)
- Max depth (~3-8)
- Learning rate (0.01-0.3)
- …

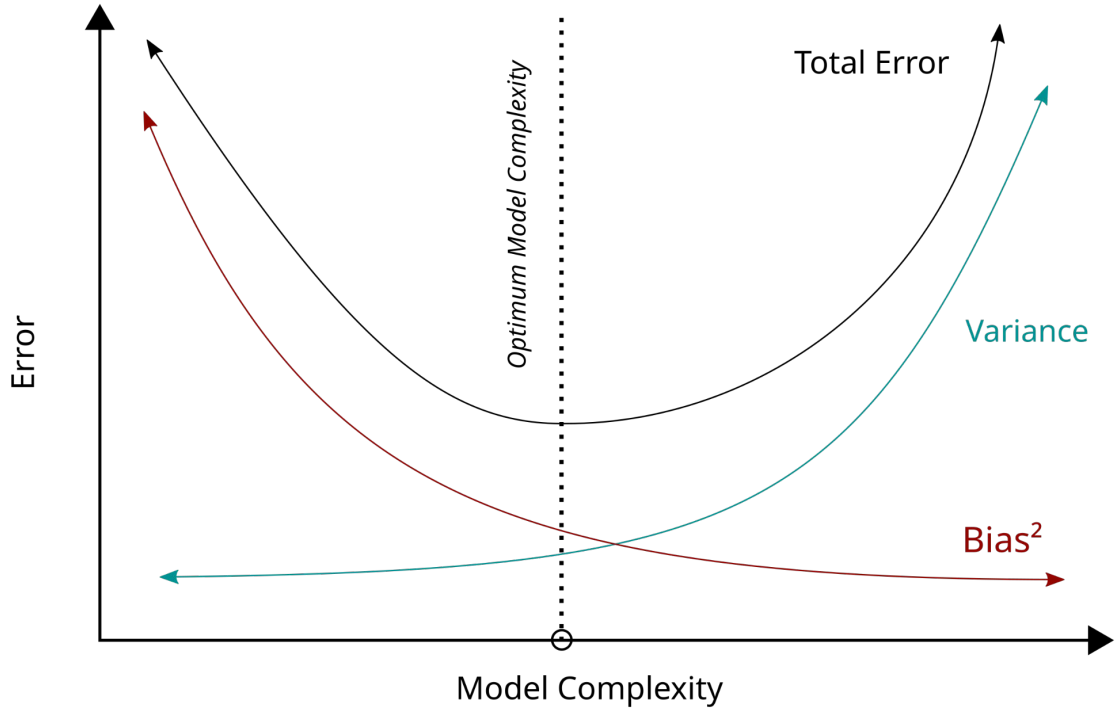
Gradient Boosting

Gradient Boosting:

- State-of-art accuracy on tabular tasks
- Can fit very complex data
- Can handle high-dimensional mixed continuous-discrete data
- Can overfit
- Sensitive to hyper-parameters

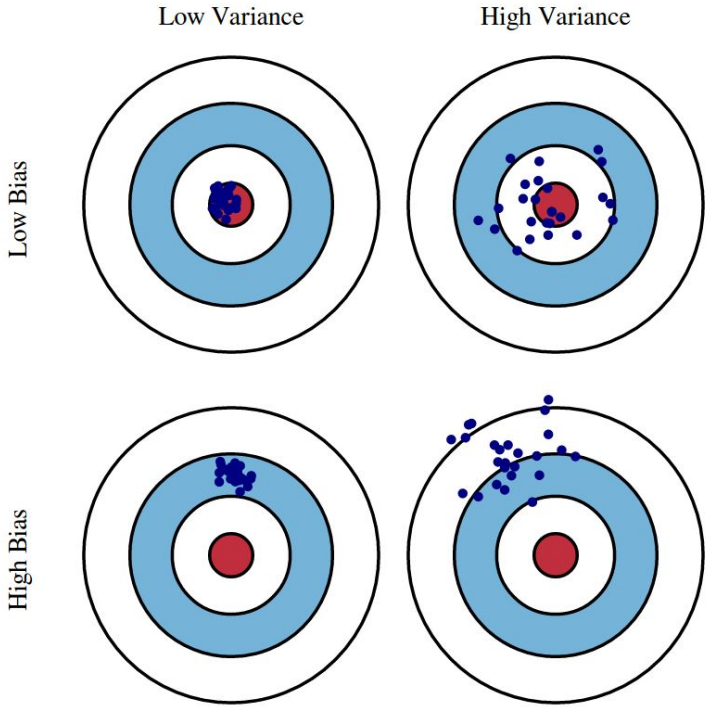

Bias-Variance
Tradeoff

Bias-Variance Tradeoff



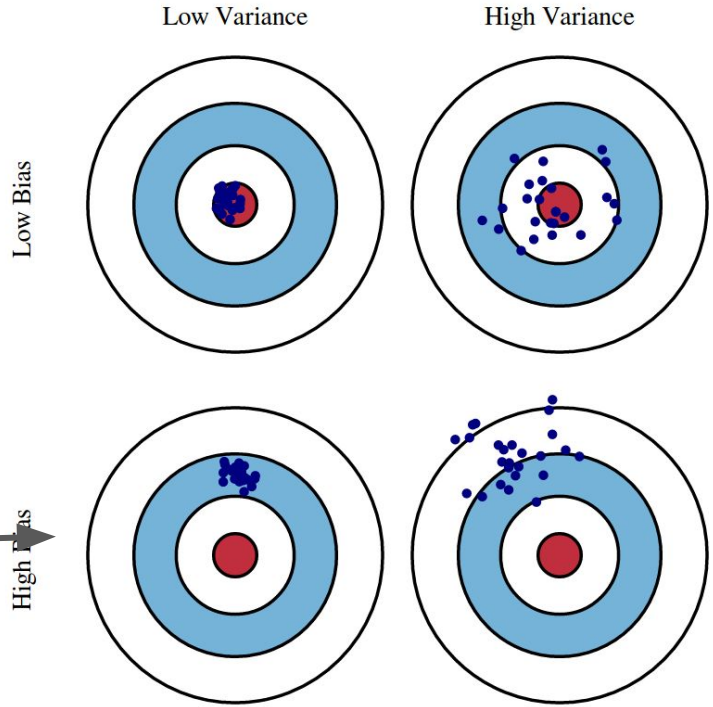
Bias variance tradeoff is closely related to underfitting vs overfitting

Bias-Variance Tradeoff



Bias variance tradeoff is closely related to underfitting vs overfitting

Bias-Variance Tradeoff



Deep Neural Net
High-depth tree

Linear model
Low-depth tree

Bias variance tradeoff is closely related to underfitting vs overfitting

Bias-Variance Decomposition

Consider a regression problem with data:

$$y = f(x) + \epsilon, \quad \mathbb{E}[\epsilon] = 0, \quad \mathrm{Var}(\epsilon) = \sigma^2$$

Let's break down the expected error, where the expectation is over

- Training data for our model $\hat{f}(x)$
- Observation noise ϵ at our test datapoint x

$$\mathbb{E}[(y - \hat{f}(x))^2] =$$

Bias-Variance Decomposition

Consider a regression problem with data:

$$y = f(x) + \epsilon, \quad \mathbb{E}[\epsilon] = 0, \quad \mathrm{Var}(\epsilon) = \sigma^2$$

Let's break down the expected error, where the expectation is over

- Training data for our model $\hat{f}(x)$
- Observation noise ϵ at our test datapoint x

$$\begin{aligned}\mathbb{E}[(y - \hat{f}(x))^2] &= \mathbb{E}[(y - f(x) + f(x) - \hat{f}(x))^2] \\ &= \mathbb{E}[(y - f(x))^2 + 2(y - f(x))(f(x) - \hat{f}(x)) + (f(x) - \hat{f}(x))^2]\end{aligned}$$

Bias-Variance Decomposition

$$\begin{aligned}\mathbb{E}[(y-\hat{f}(x))^2] &= \mathbb{E}[(y-f(x)+f(x)-\hat{f}(x))^2] \\ &= \mathbb{E}[(y-f(x))^2 + 2(y-f(x))(f(x)-\hat{f}(x)) + (f(x)-\hat{f}(x))^2]\end{aligned}$$

$$\begin{aligned}\mathbb{E}[(y-f(x))^2] &= \mathbb{E}[\epsilon^2] = \sigma^2 \\ \mathbb{E}[2(y-f(x))(f(x)-\hat{f}(x))] &= 2\mathbb{E}[\epsilon]\mathbb{E}[f(x)-\hat{f}(x)] = 0 \\ \mathbb{E}[(f(x)-\hat{f}(x))^2] &= (\text{to be decomposed})\end{aligned}$$

Bias-Variance Decomposition

$$\begin{aligned}\mathbb{E}[(y-\hat{f}(x))^2] &= \mathbb{E}[(y-f(x)+f(x)-\hat{f}(x))^2] \\ &= \mathbb{E}[(y-f(x))^2 + 2(y-f(x))(f(x)-\hat{f}(x)) + (f(x)-\hat{f}(x))^2]\end{aligned}$$

$$\begin{aligned}\mathbb{E}[(y-f(x))^2] &= \mathbb{E}[\epsilon^2] = \sigma^2 \\ \mathbb{E}[2(y-f(x))(f(x)-\hat{f}(x))] &= 2\mathbb{E}[\epsilon]\mathbb{E}[f(x)-\hat{f}(x)] = 0 \\ \mathbb{E}[(f(x)-\hat{f}(x))^2] &= (\text{to be decomposed})\end{aligned}$$

Bias-Variance Decomposition

Now, the expectation is only over the training data for our model

$$
\begin{aligned}
\mathbb{E}[(f(x) - \hat{f}(x))^2] &= \mathbb{E}[(f(x) - \mathbb{E}[\hat{f}(x)] + \mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2] \\
&= \mathbb{E}[(f(x) - \mathbb{E}[\hat{f}(x)])^2 + 2(f(x) - \mathbb{E}[\hat{f}(x)])(\mathbb{E}[\hat{f}(x)] - \hat{f}(x)) \\
&\quad + (\mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2] \\
&= (f(x) - \mathbb{E}[\hat{f}(x)])^2 + 0 + \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2] \\
&= \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x))
\end{aligned}
$$

Bias-Variance Decomposition

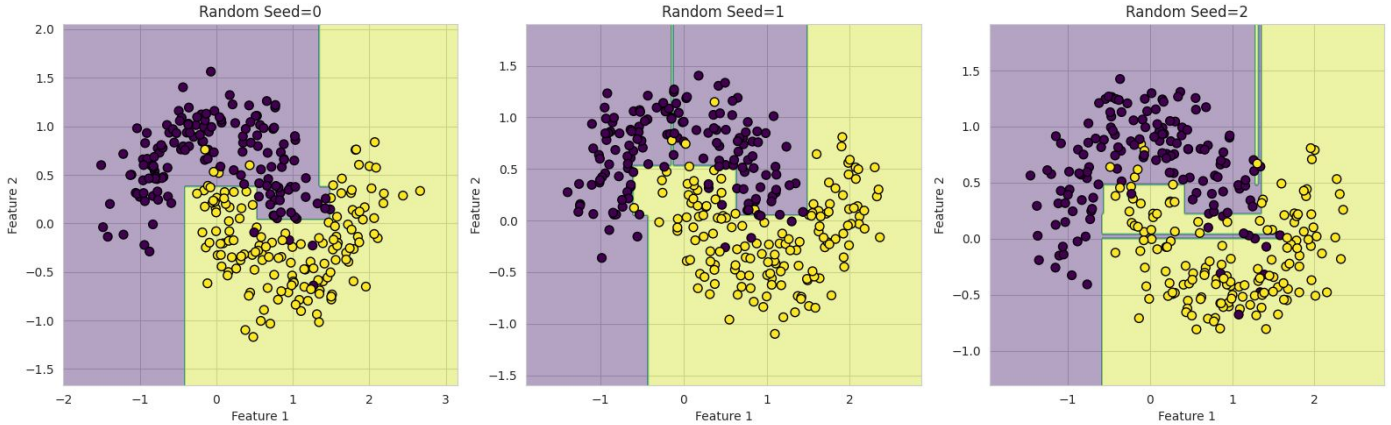
$$\boxed{\mathbb{E}[(y - \hat{f}(x))^2] = \sigma^2 + \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x))}$$

where:

- $\sigma^2 = \text{Var}(\epsilon)$ is the irreducible error (noise variance)
- $\text{Bias}(\hat{f}(x)) = \mathbb{E}[\hat{f}(x)] - f(x)$ measures how far the average prediction is from the true function
- $\text{Var}(\hat{f}(x)) = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$ measures the variability of predictions across different training sets

Bias-Variance Decomposition

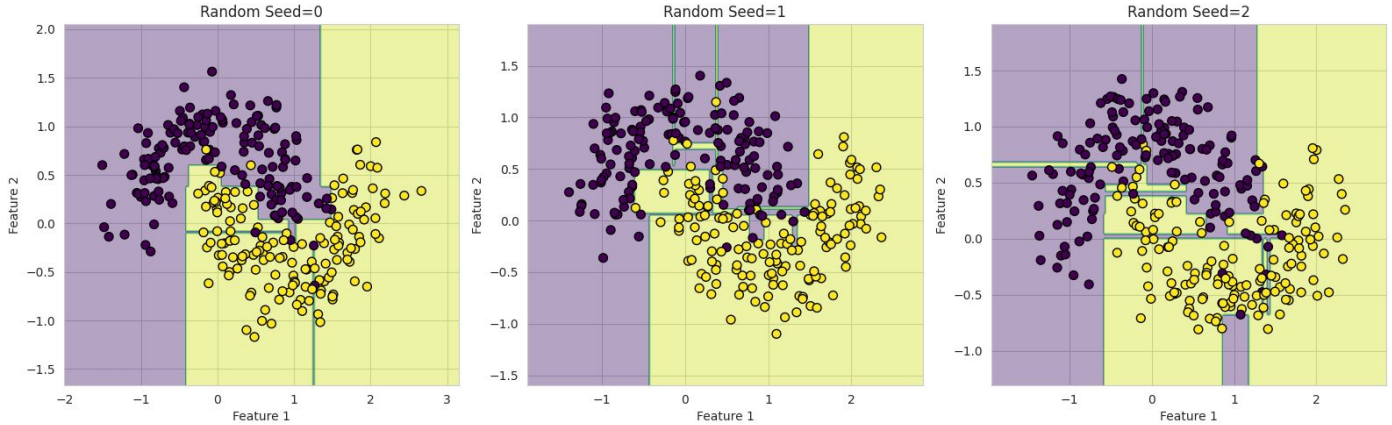
$$\mathbb{E}[(y - \hat{f}(x))^2] = \sigma^2 + \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x))$$



High-depth decision trees are high-variance, low-bias.

Bias-Variance Decomposition

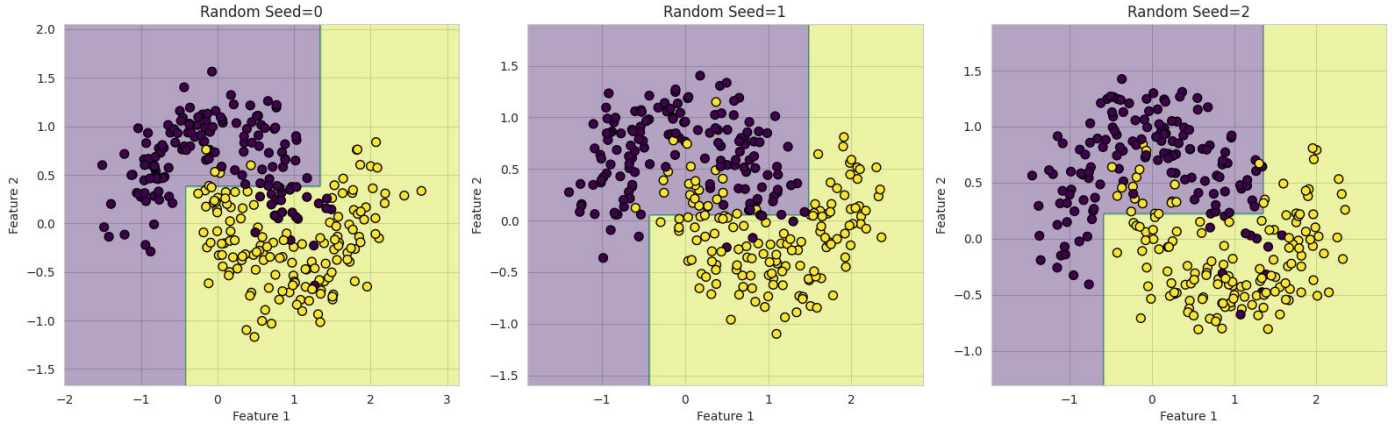
$$\mathbb{E}[(y - \hat{f}(x))^2] = \sigma^2 + \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x))$$



High-depth (10) decision trees are high-variance, low-bias.

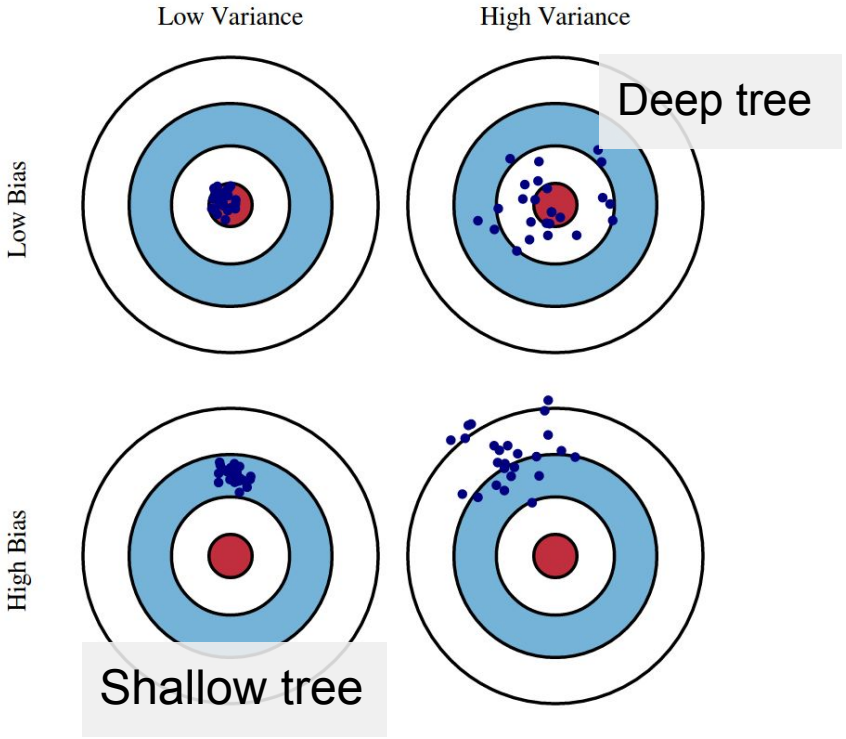
Bias-Variance Decomposition

$$\mathbb{E}[(y - \hat{f}(x))^2] = \sigma^2 + \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x))$$



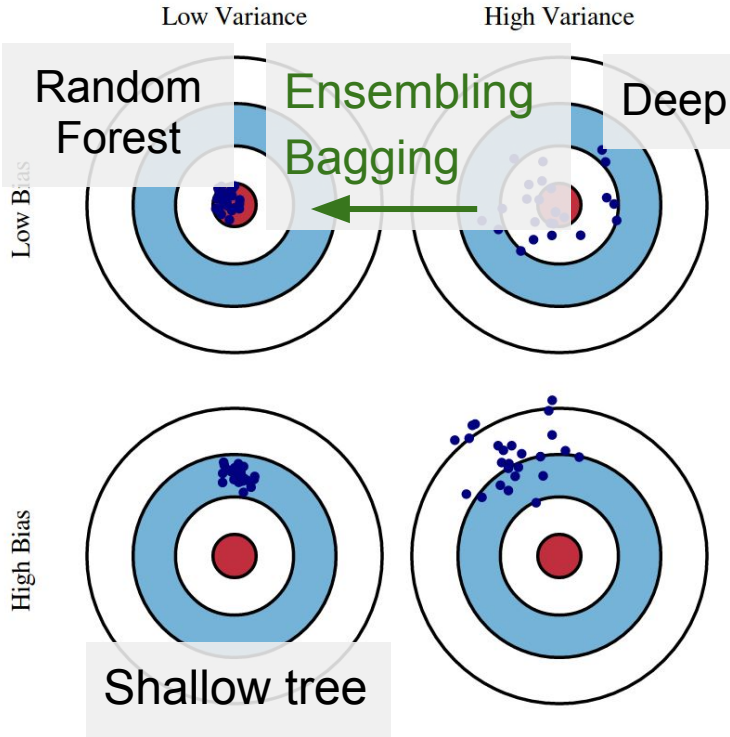
Low-depth (2) decision trees are low-variance, high-bias.

Bias-Variance Decomposition



$$\mathbb{E}[(y - \hat{f}(x))^2] = \sigma^2 + \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x))$$

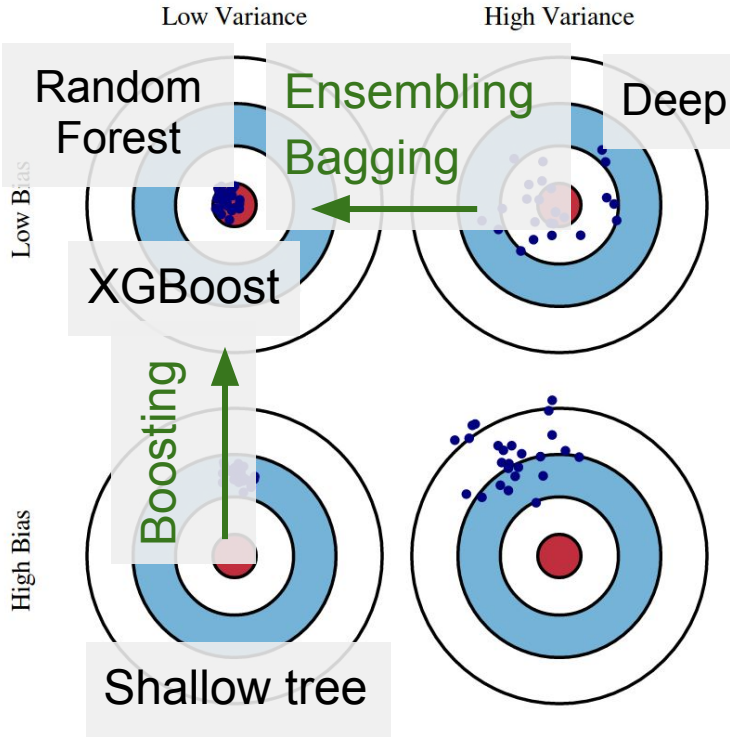
Bias-Variance Decomposition



Bagging and ensembling generally reduce variance

$$\mathbb{E}[(y - \hat{f}(x))^2] = \sigma^2 + \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x))$$

Bias-Variance Decomposition



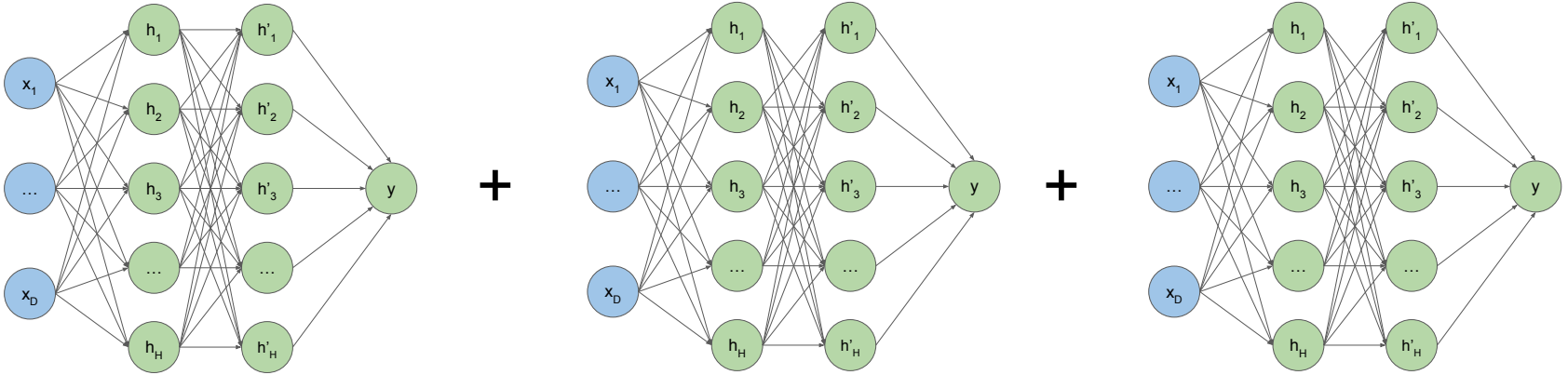
Bagging and ensembling generally reduce variance

Boosting reduces bias

$$\mathbb{E}[(y - \hat{f}(x))^2] = \sigma^2 + \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x))$$

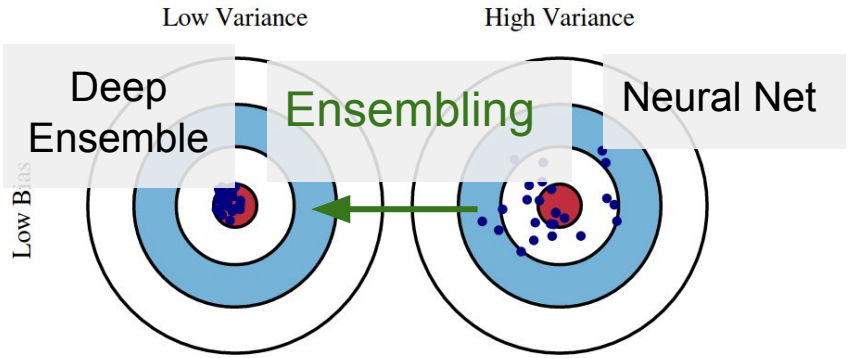

Deep
Ensembles

Deep Ensembles



- Train *N* copies of the neural network with different weights
- Use different random initializations
- Average predictions
- Better accuracy and *uncertainty*

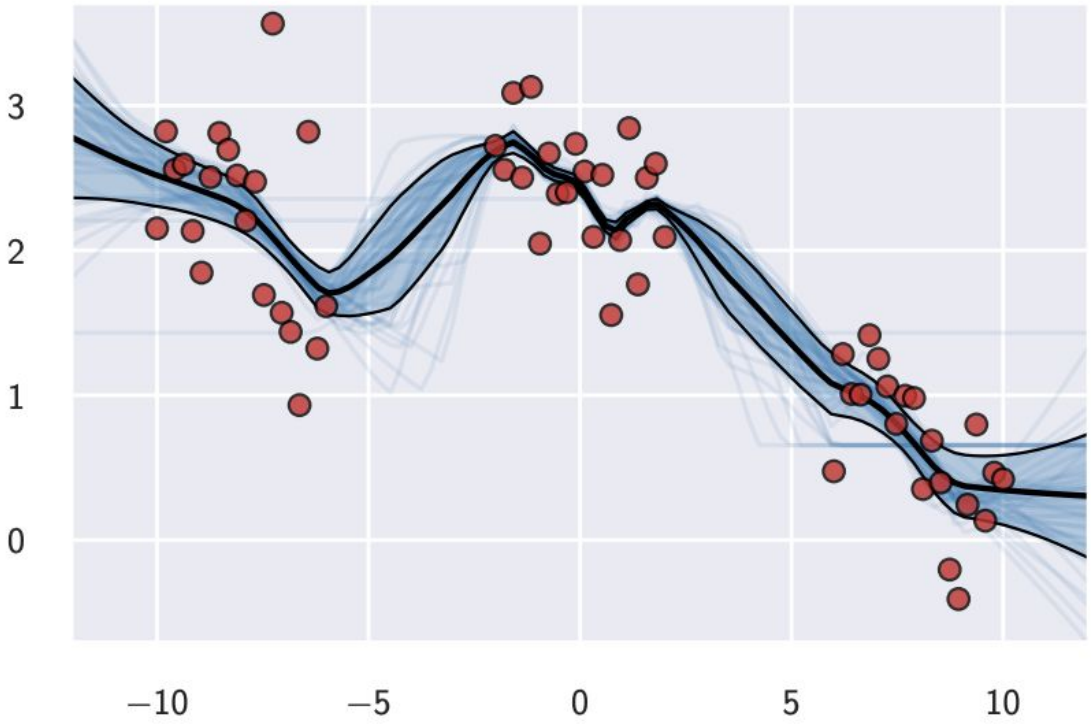
Deep Ensembles



Neural nets are flexible ⇒ high variance, low bias.

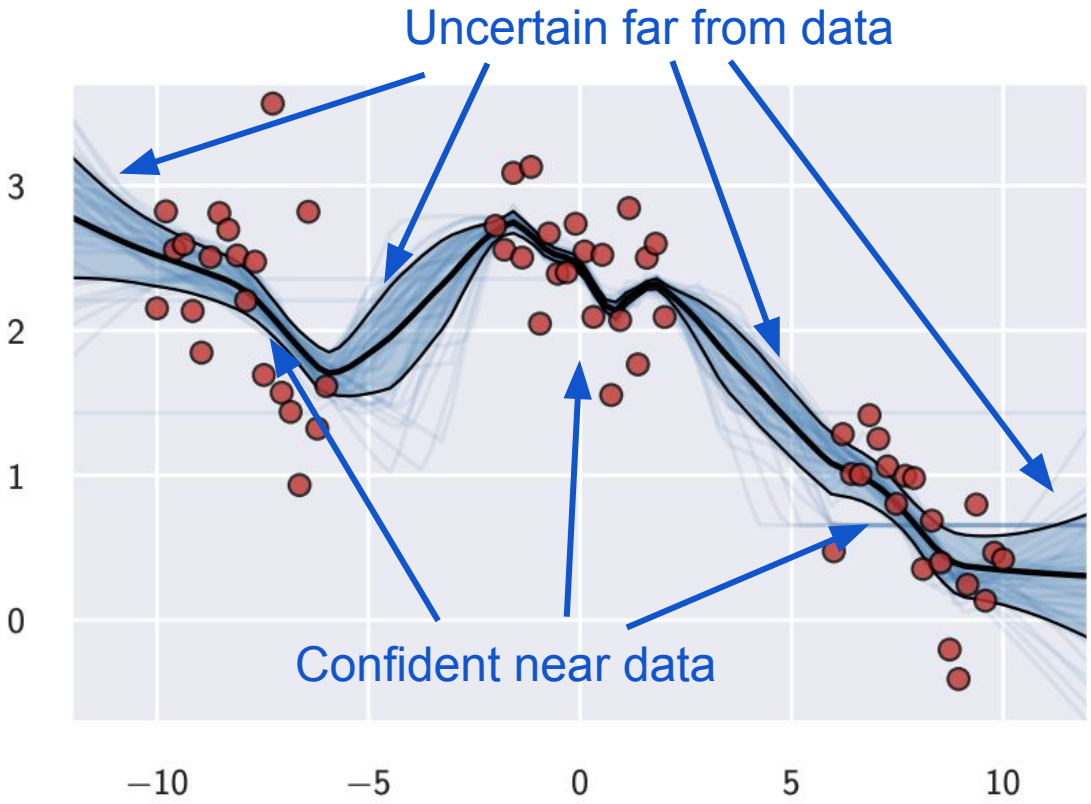
Unlike Random Forest, we don’t usually randomize data or features in Deep Ensembles.

Uncertainty



In addition to accurate predictions, we often want to know how confident we should be.

Uncertainty



In addition to accurate predictions, we often want to know how confident we should be.

Ensembling helps with uncertainty estimation.

Summary

- New model for regression and classification: decision trees
  - Interpretable, fast
  - Unstable, likes to overfit
- Ensembling: average multiple models ⇒ Random Forests
- Boosting: sequentially fit models to improve prediction ⇒ Boosting
- Ensembling can be applied to any models, including neural nets
  - Better predictions + Uncertainty