Decision Trees & Ensemble Models

Panos Parpas, CQF, January 2023

Contents and Aims

- Introduce Decision Trees
- **CART**: Classification and Regression Trees
- Explain the **pros/cons** of decision trees
- Bias/Variance **Tradeoffs**
- **Ensemble learning** (Bagging, Random Forests)
- **Boosting**: AdaBoost, Gradient Boosting
- **Applications** to finance

Problem: Which character from Asterix would default on their loan?

Falbala

Asterix

Getafix

Cacofonix

Karabella

Obelix

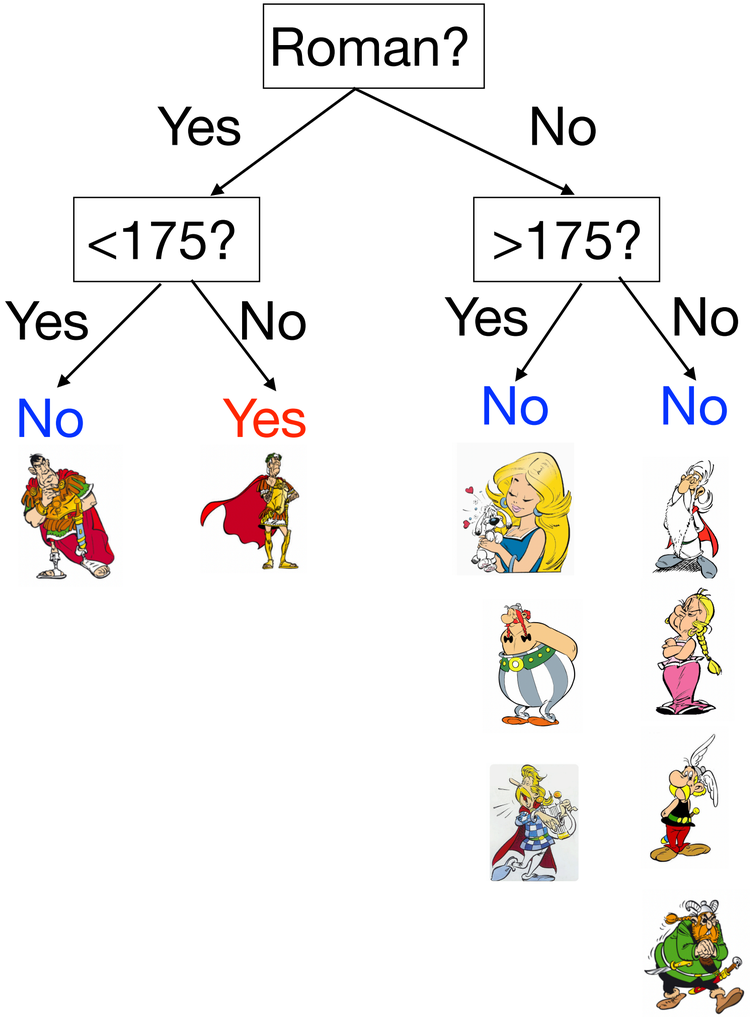
Brutus

Julius Ceasar

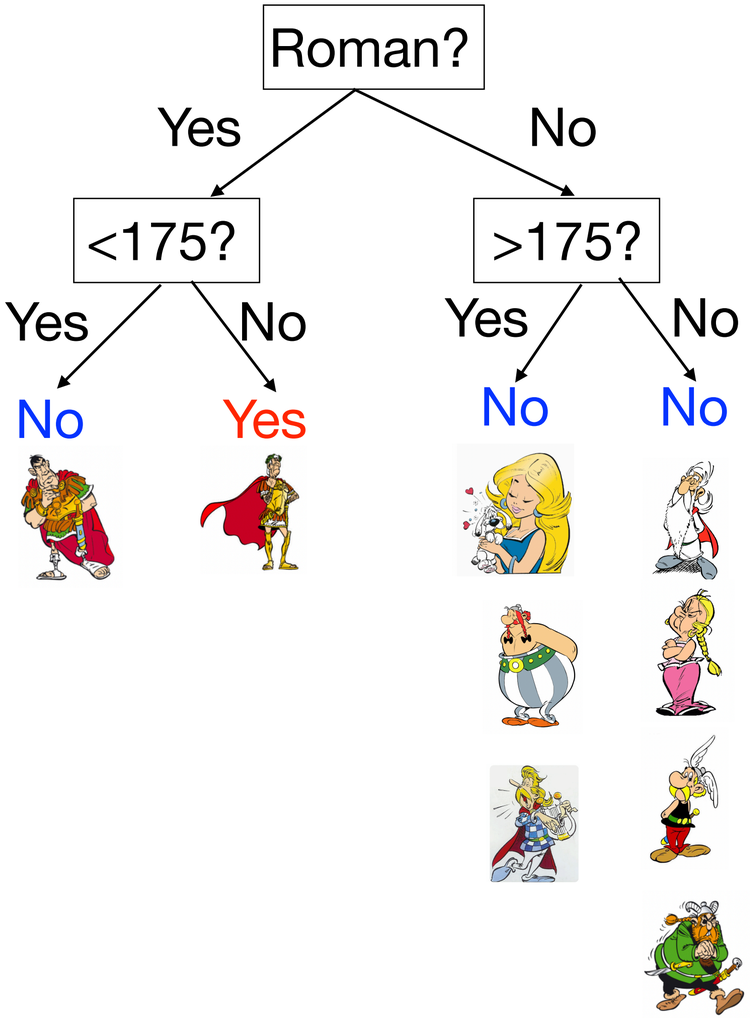
Barbe Rouge

Pegleg

	Hair	Beard	Citizenship	Helmet	Height	Default?
 Falbala	Long	N	Gaul	N	176	No
 Asterix	Short	N	Gaul	Y	155	No
 Getafix	Short	Y	Gaul	N	150	Yes
 Karabella	Long	N	Gaul	N	150	No
 Obelix	Long	N	Gaul	Y	180	Yes
 Julius Ceasar	Short	N	Roman	N	177	Yes
 Brutus	Short	N	Roman	N	170	No
 Barbe Rouge	Short	Y	Pirate	Y	171	Yes
 Cacofonix	Short	N	Gaul	N	178	No
 Pegleg	Short	N	Pirate	Y	162	?

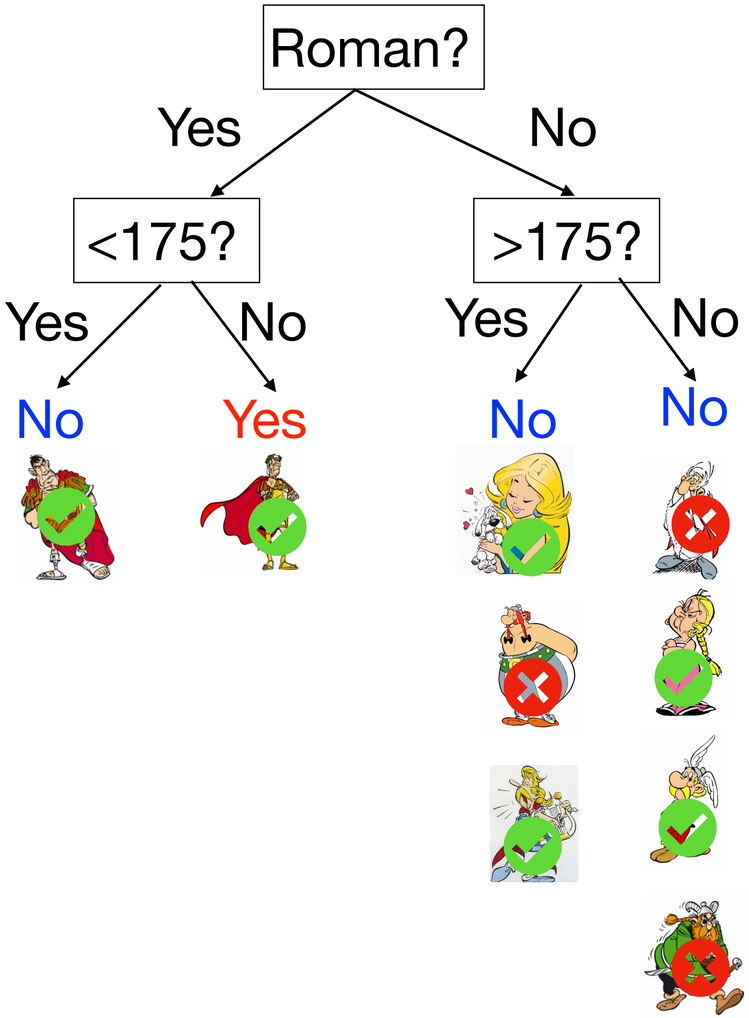


	Hair	Beard	Citizenship	Helmet	Height	Default?
 Falbala	Long	N	Gaul	N	176	No
 Asterix	Short	N	Gaul	Y	155	No
 Getafix	Short	Y	Gaul	N	150	Yes
 Karabella	Long	N	Gaul	N	150	No
 Obelix	Long	N	Gaul	Y	180	Yes
 Julius Ceasar	Short	N	Roman	N	177	Yes
 Brutus	Short	N	Roman	N	170	No
 Barbe Rouge	Short	Y	Pirate	Y	171	Yes
 Cacofonix	Short	N	Gaul	N	178	No
 Pegleg	Short	N	Pirate	Y	162	?



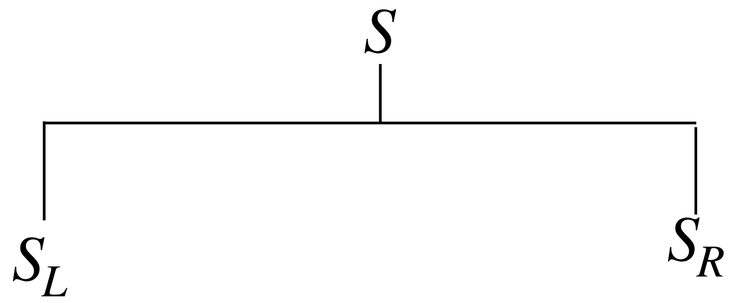
Is this a good tree?

	Hair	Beard	Citizenship	Helmet	Height	Default?
 Falbala	Long	N	Gaul	N	176	No
 Asterix	Short	N	Gaul	Y	155	No
 Getafix	Short	Y	Gaul	N	150	Yes
 Karabella	Long	N	Gaul	N	150	No
 Obelix	Long	N	Gaul	Y	180	Yes
 Julius Ceasar	Short	N	Roman	N	177	Yes
 Brutus	Short	N	Roman	N	170	No
 Barbe Rouge	Short	Y	Pirate	Y	171	Yes
 Cacofonix	Short	N	Gaul	N	178	No
 Pegleg	Short	N	Pirate	Y	162	?



CART — Classification and Regression Trees

Input: $S = (x_i, y_i),\ i = 1, \ldots, N$



1. Partition the original region into a tree
2. Split the tree to improve the “quality” of the tree
3. Compute the prediction of each leaf: $\kappa(m) = \arg\max_k p_{mk}$

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{I}(y_i = k)$$

Leaf node m with data points R_m and N_m number of observations in region m

Example

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{I}(y_i = k) \quad \kappa(m) = \arg\max_k p_{mk}$$

Leaf node m with data points R_m and

N_m number of observations in region m

$k = 2$ (number of classes)

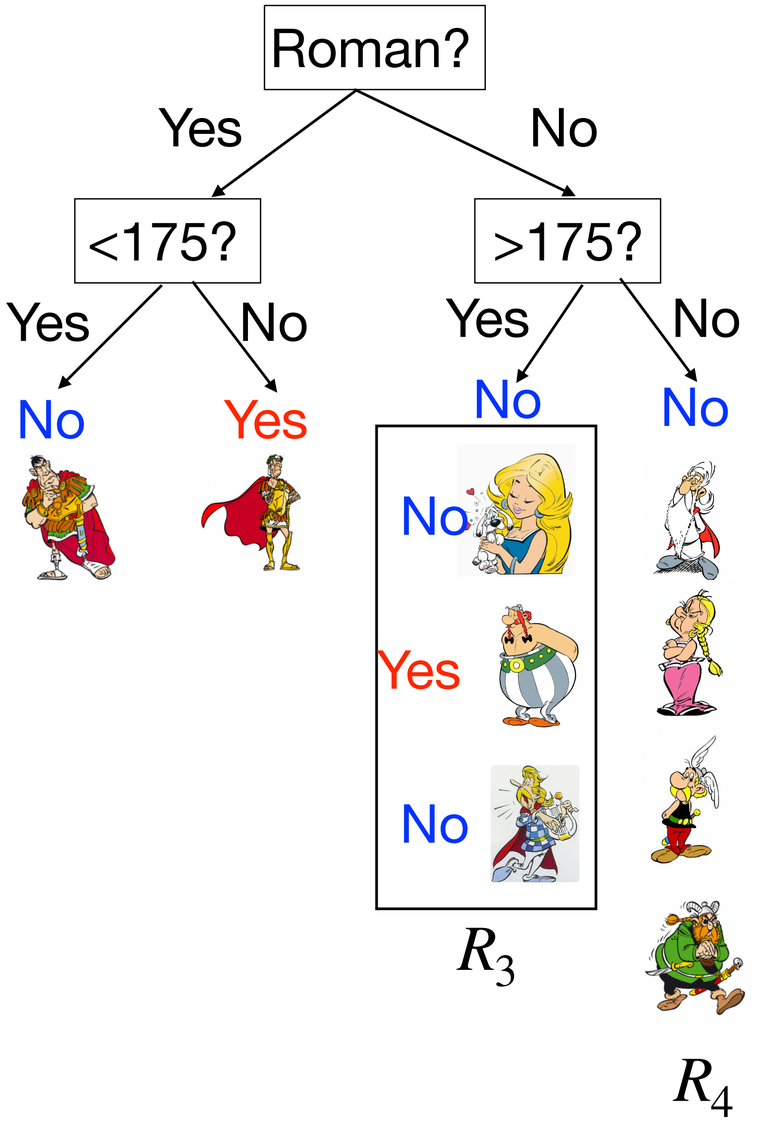
$m = 3$ (Region/Node 3, R_3)

$N_3 = 3$ (members in R_3)

$p_{3,0} = \frac{2}{3}$ (members in R_3 no default)

$p_{3,1} = \frac{1}{3}$ (members in R_3 default)

$\kappa_4 = 0$ (most likely class: No)



Example

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{I}(y_i = k) \quad \kappa(m) = \arg\max_k p_{mk}$$

Leaf node m with data points R_m and

N_m number of observations in region m

$k = 2$ (number of classes)

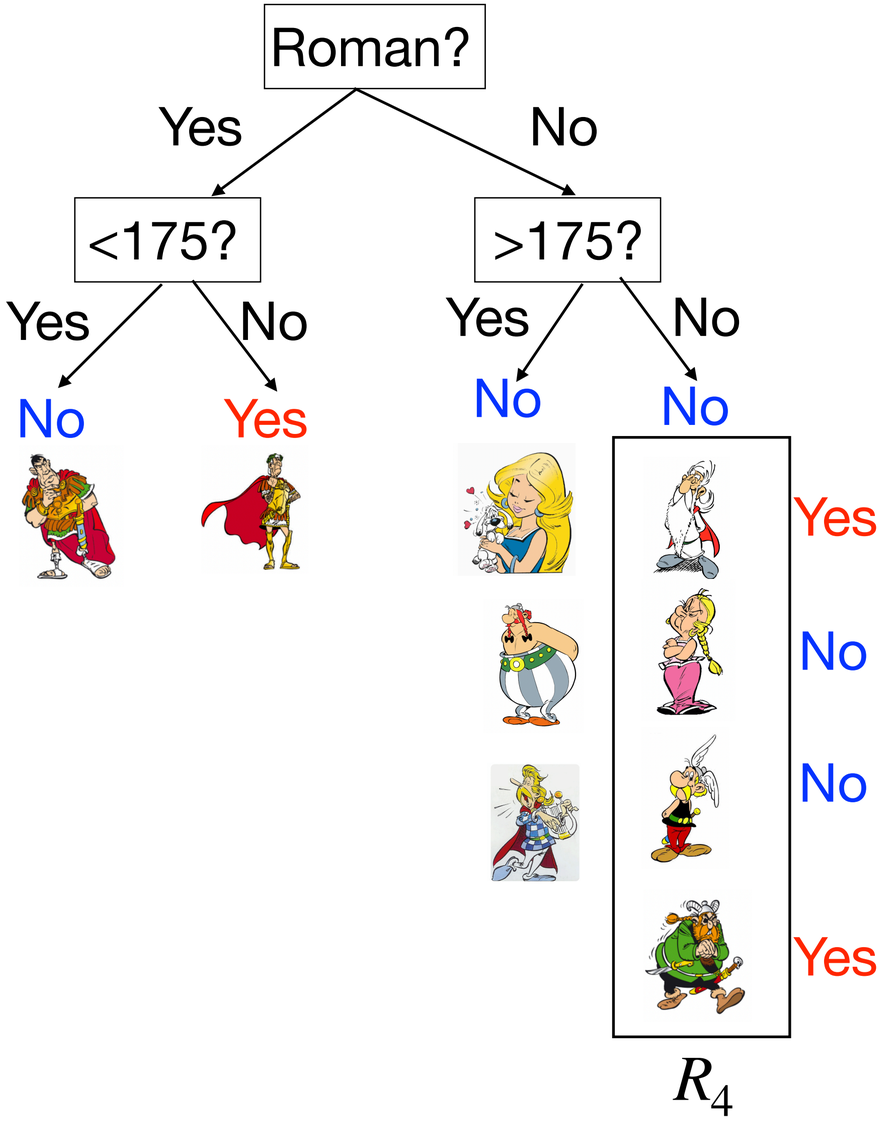
$m = 4$ (Region/Node 4, R_4)

$N_4 = 4$ (members in R_4)

$p_{4,0} = \frac{2}{4}$ (members in R_4 no default)

$p_{4,1} = \frac{2}{4}$ (members in R_4 default)

$\kappa_4 = 0$ (break tie randomly)



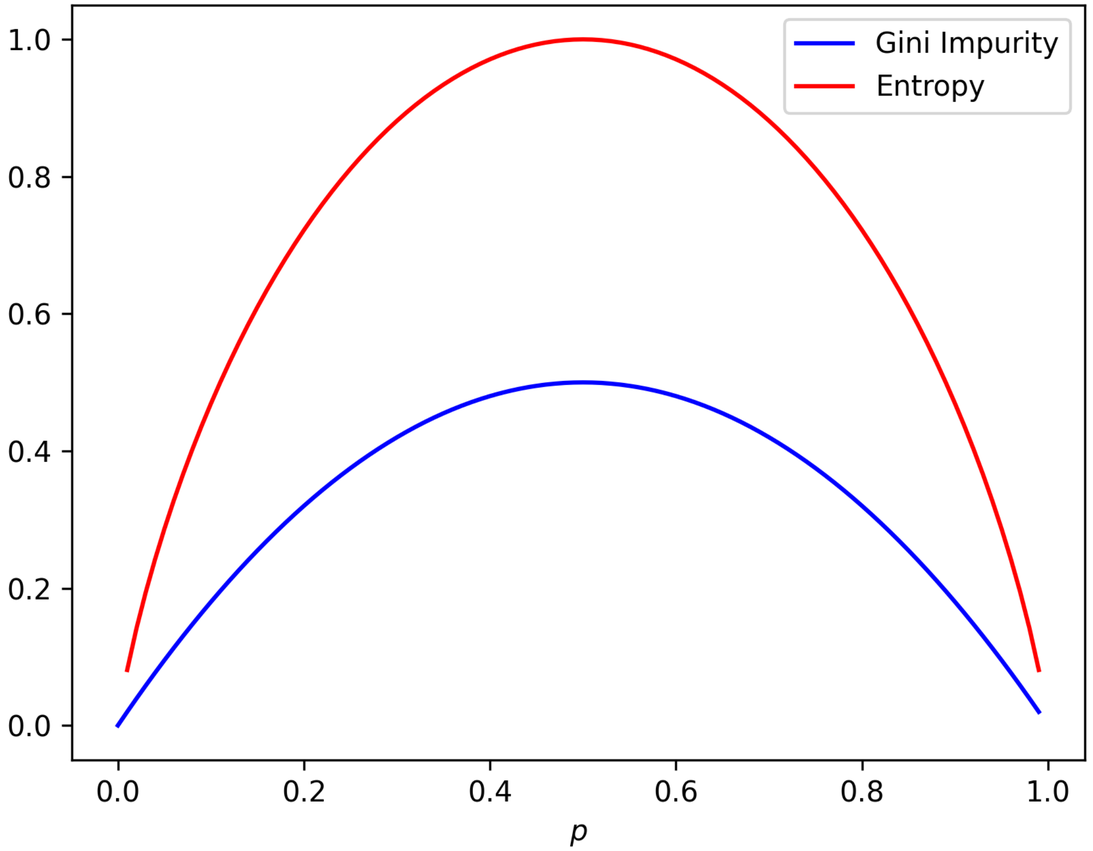
CART — Classification and Regression Trees

What is a good tree?

1. Good trees are pure i.e. each leaf contains a single class.
2. Bad trees have a lot of uncertainty.

Gini impurity index: $\sum_{k=1}^{K} p_{mk}(1 - p_{mk})$

Negative Entropy: $-\sum_{k=1}^{K} p_{mk} \log(p_{mk})$



Example

Gini impurity index: $\sum_{k=1}^{K} p_{mk}(1-p_{mk})$

$k = 2$ (number of classes)

$N = 9$ (number of data points)

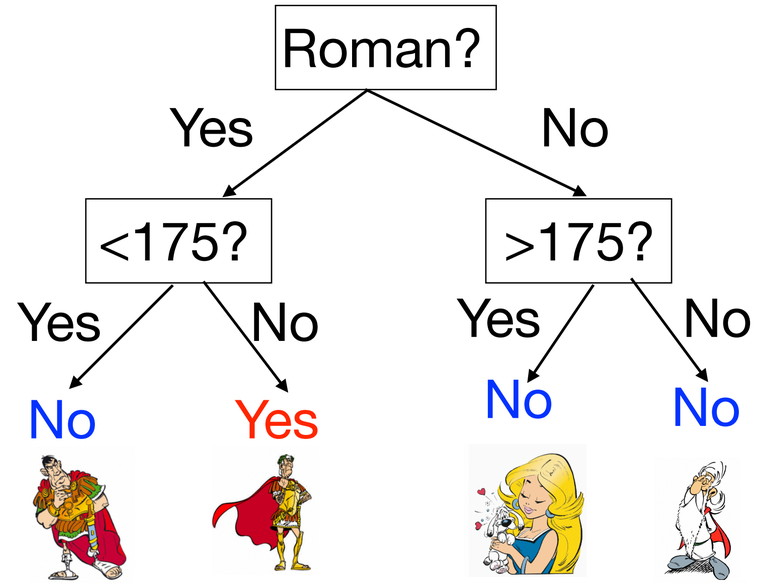
$$GI_1 = 0(1-0) + 1(1-1) = 0$$

$$GI_2 = 1(1-1) + 0(1-0) = 0$$

$$GI_3 = \frac{2}{3}\frac{1}{3} + \frac{1}{3}\frac{2}{3} = \frac{4}{9}$$

$$GI_4 = \frac{1}{2}\frac{1}{2} + \frac{1}{2}\frac{1}{2} = \frac{1}{2}$$

$$GI = \frac{1}{9}GI_1 + \frac{1}{9}GI_2 + \frac{3}{9}GI_3 + \frac{4}{9}GI_4$$



$$p_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad p_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

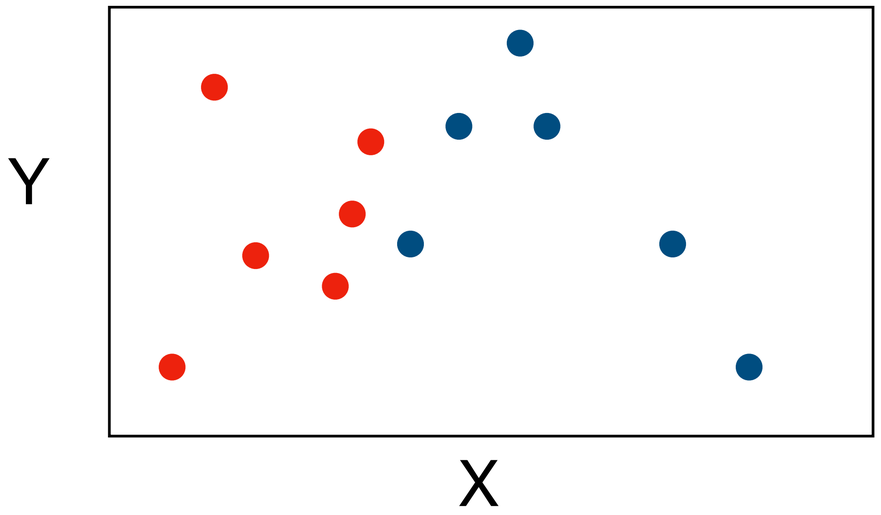
$$p_3 = \begin{bmatrix} 1/3 \\ 2/3 \end{bmatrix}$$

$$p_4 = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}$$

CART — Classification and Regression Trees

Objective: Find the tree with the lowest impurity index.

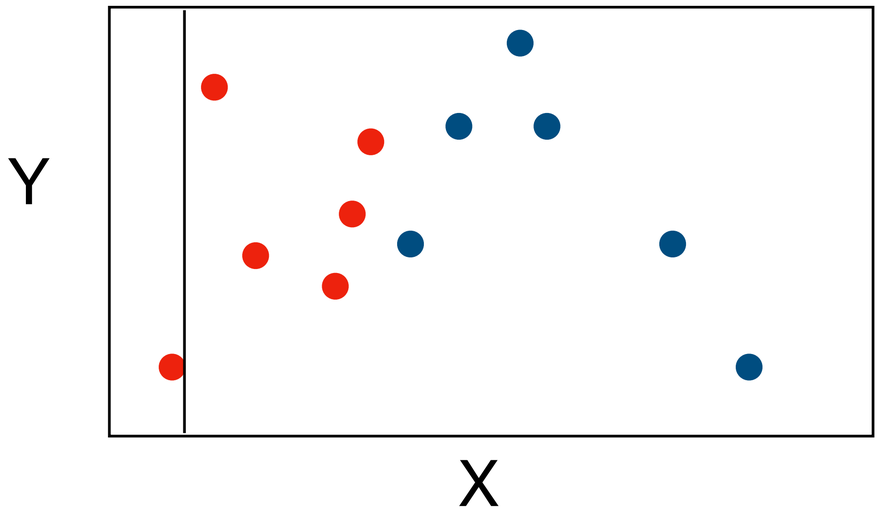
Hard problem, use heuristics instead: Try all possible splits



CART — Classification and Regression Trees

Objective: Find the tree with the lowest impurity index.

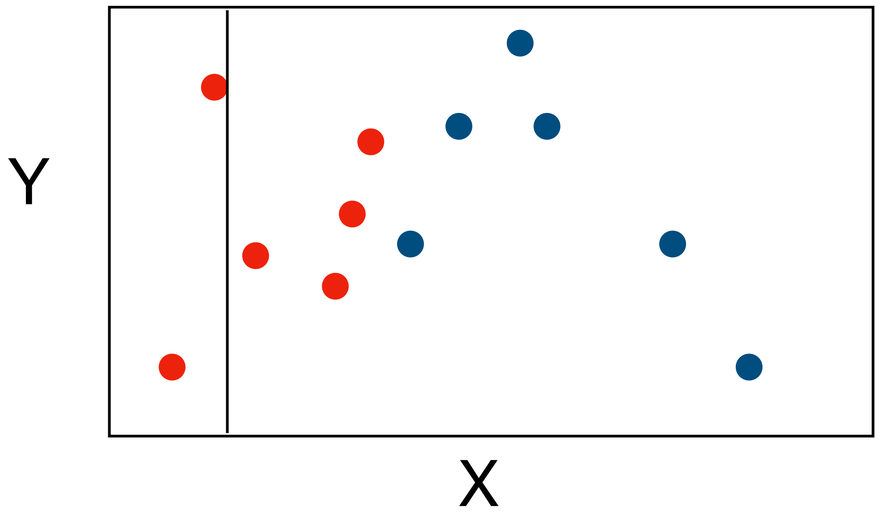
Hard problem, use heuristics instead: Try all possible splits



CART — Classification and Regression Trees

Objective: Find the tree with the lowest impurity index.

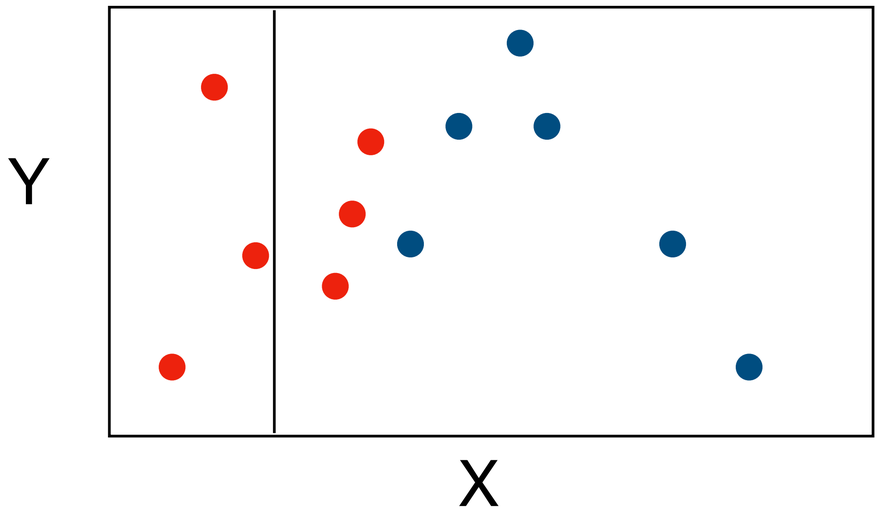
Hard problem, use heuristics instead: Try all possible splits



CART — Classification and Regression Trees

Objective: Find the tree with the lowest impurity index.

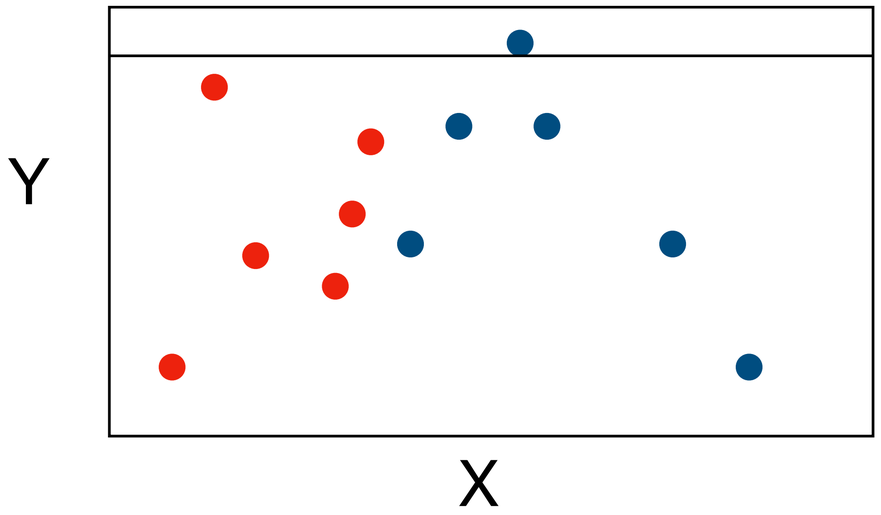
Hard problem, use heuristics instead: Try all possible splits



CART — Classification and Regression Trees

Objective: Find the tree with the lowest impurity index.

Hard problem, use heuristics instead: Try all possible splits



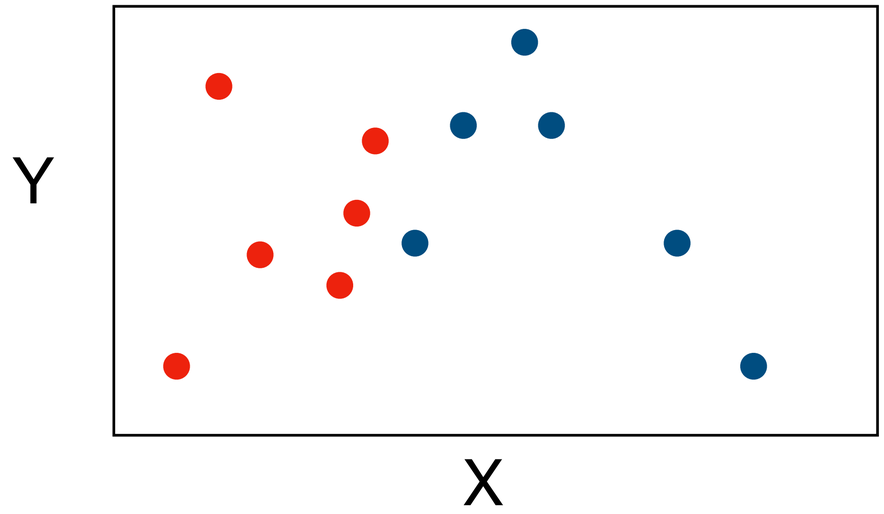
CART — Classification and Regression Trees

Objective: Find the tree with the lowest impurity index.

Hard problem, use heuristics instead: Try all possible splits

Compute the impurity of each split and pick the best one!

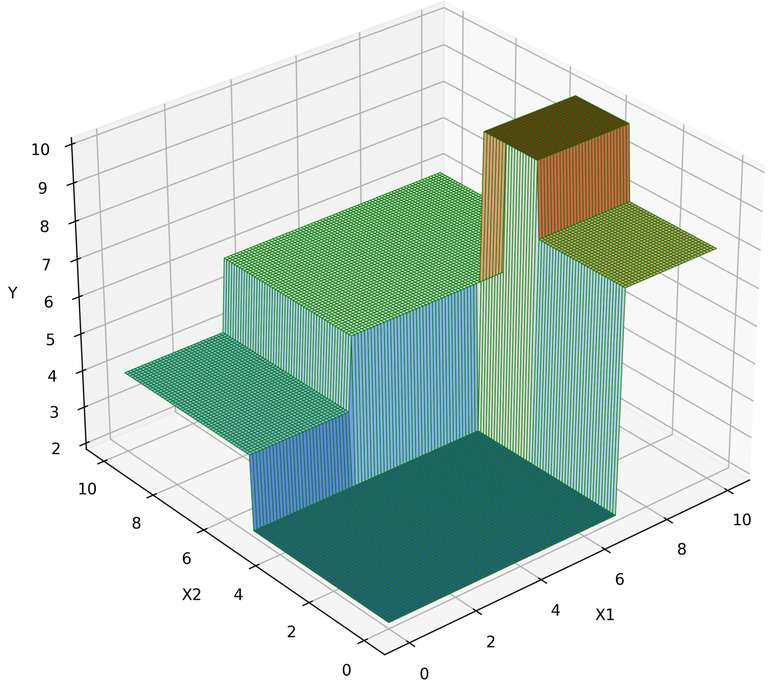
Is this optimal? **No,** but it can be computed in complexity linear in feature dimensions and data points



CART — Classification and Regression Trees

Input: $(x_i, y_i),\ i = 1, \ldots, N$

1. Partition the original region into a tree
2. Associate mean response with each region e.g.



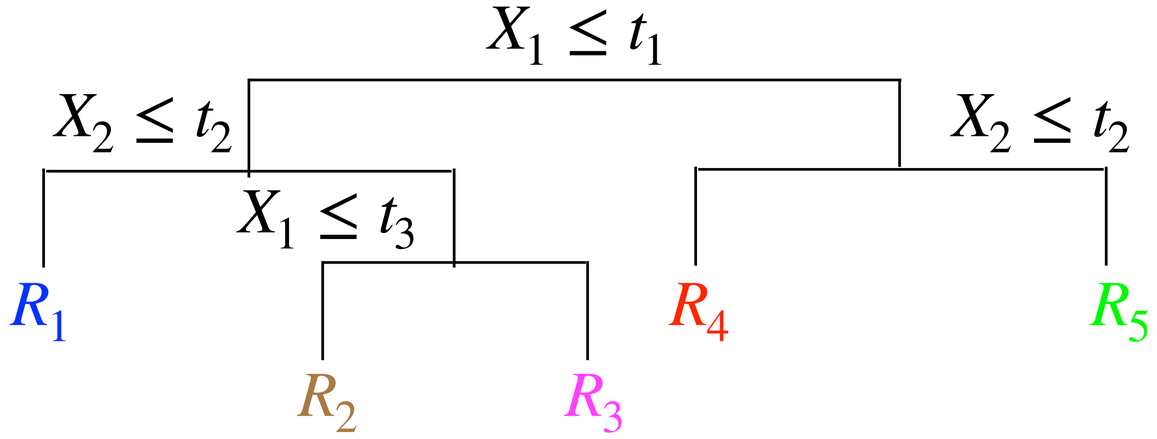
$$R_1 = \{x \mid x_1 \leq t_1,\ x_2 \leq t_2\}$$ **Region 1**

$$w_1 = \frac{\sum_{i=1}^{N} y_i \mathbb{I}(x^i \in R_1)}{\sum_{i=1}^{N} \mathbb{I}(x^i \in R_1)}$$

$x^i = (x_1^i, x_2^i)$ I-th point

Mean response for Region 1, where

$$I(x \in R) = \begin{cases} 1 \text{ if } x \in R \\ 0 \text{ otherwise} \end{cases}$$



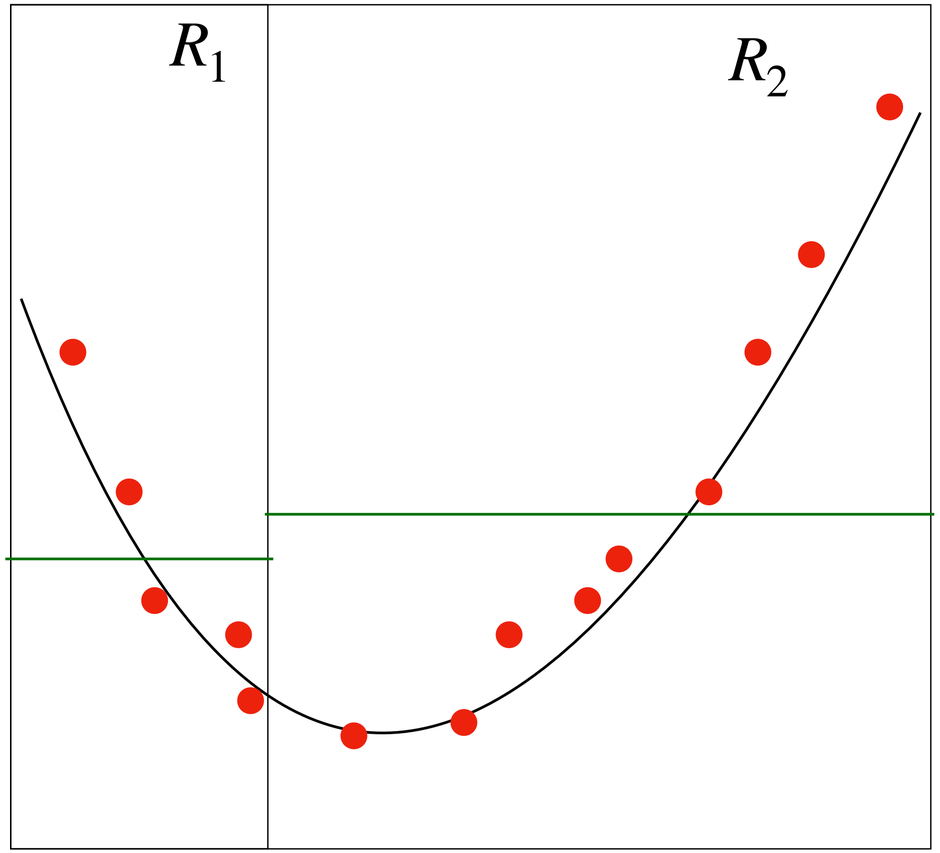
Final output:

$$f(x; R_1, \ldots, R_J) = \sum_{j=1}^{J} w_j \mathbb{I}(x \in R_j)$$

Model Fitting

Output of regression tree:

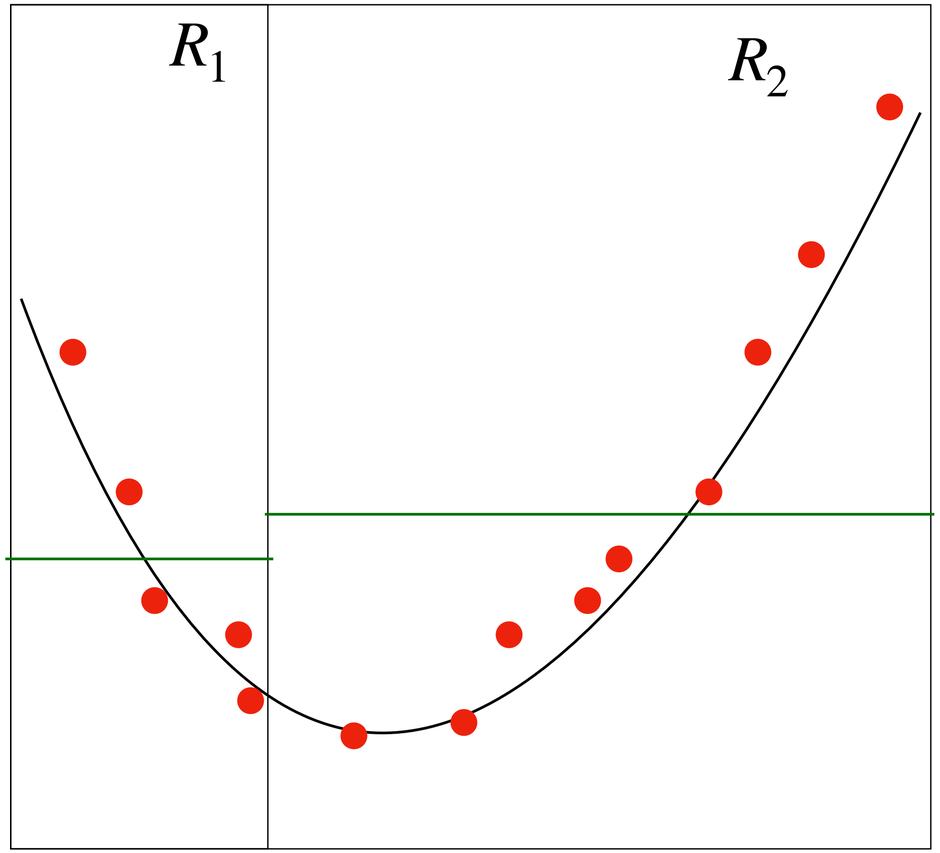
$$f(x; R_1, \ldots, R_J) = \sum_{j=1}^{J} w_j \mathbb{I}(x \in R_j) = \text{ave. of all y's in region where x belongs}$$



Model Fitting

Output of regression tree:

$$f(x; R_1, \ldots, R_J) = \sum_{j=1}^{J} w_j \mathbb{I}(x \in R_j) = \text{ave. of all y's in region where x belongs}$$



compute the error associated with each split e.g.

$$\sum_{i=1}^{N} (y_i - f(x; R_{1,l}, R_{1,r}, R_2, \ldots R_J))^2$$

Repeat for all regions and splits and choose the best one

Instead of impurity, select the tree with the smallest variance

Regularisation for Decision Trees

If a tree is allowed to grow with no constraints, then it can achieve zero training error but it will **memorise the training data set and overfit**

Simple regularisation techniques:

- Stop the tree growing process according to a heuristic, e.g. setting a max-depth or setting the minimum examples in a node.
- Allow the tree to go to its maximum depth and then prune it back

Decision Trees Pros

✓Trees are easy to interpret and explain

✓They can easily handle mixed, discrete and continuous data

✓They are not sensitive to scaling

✓They perform automatic variable selection

✓Robust to outliers

✓Fast and scale well

✓Can handle missing inputs

Decision Trees - Cons

Decision trees **do not work** they suffer from **extreme** bias/variance trade-off issues:

- Trees with **low depth** have **high bias** (model is too simple)
- Tree with **high depth** have **high variance** (overfit)

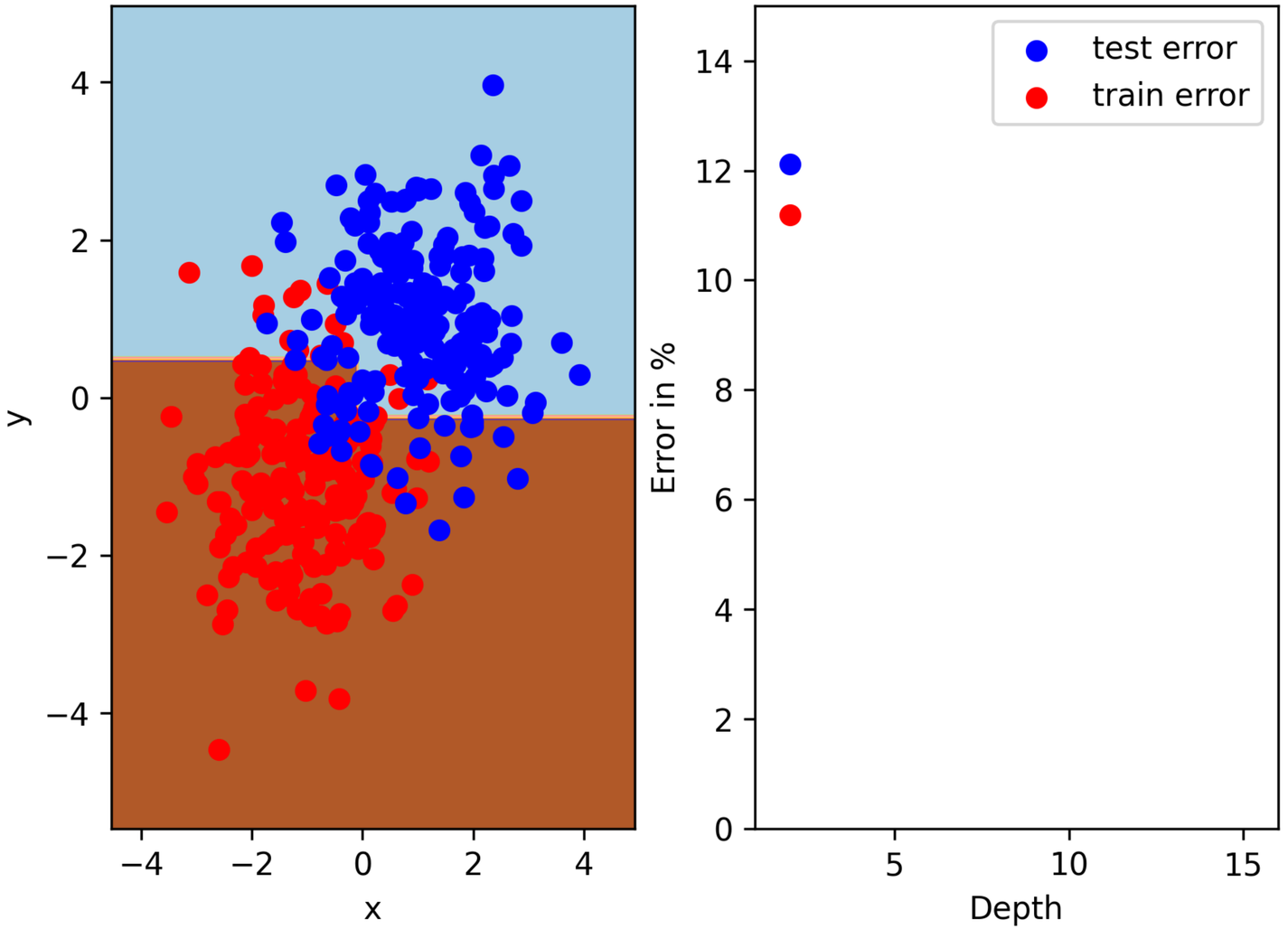
Bagging reduces variance problem

Boosting reduces bias

With these modifications, they are **state-of-the-art** (e.g. search engines are boosted decision trees!)

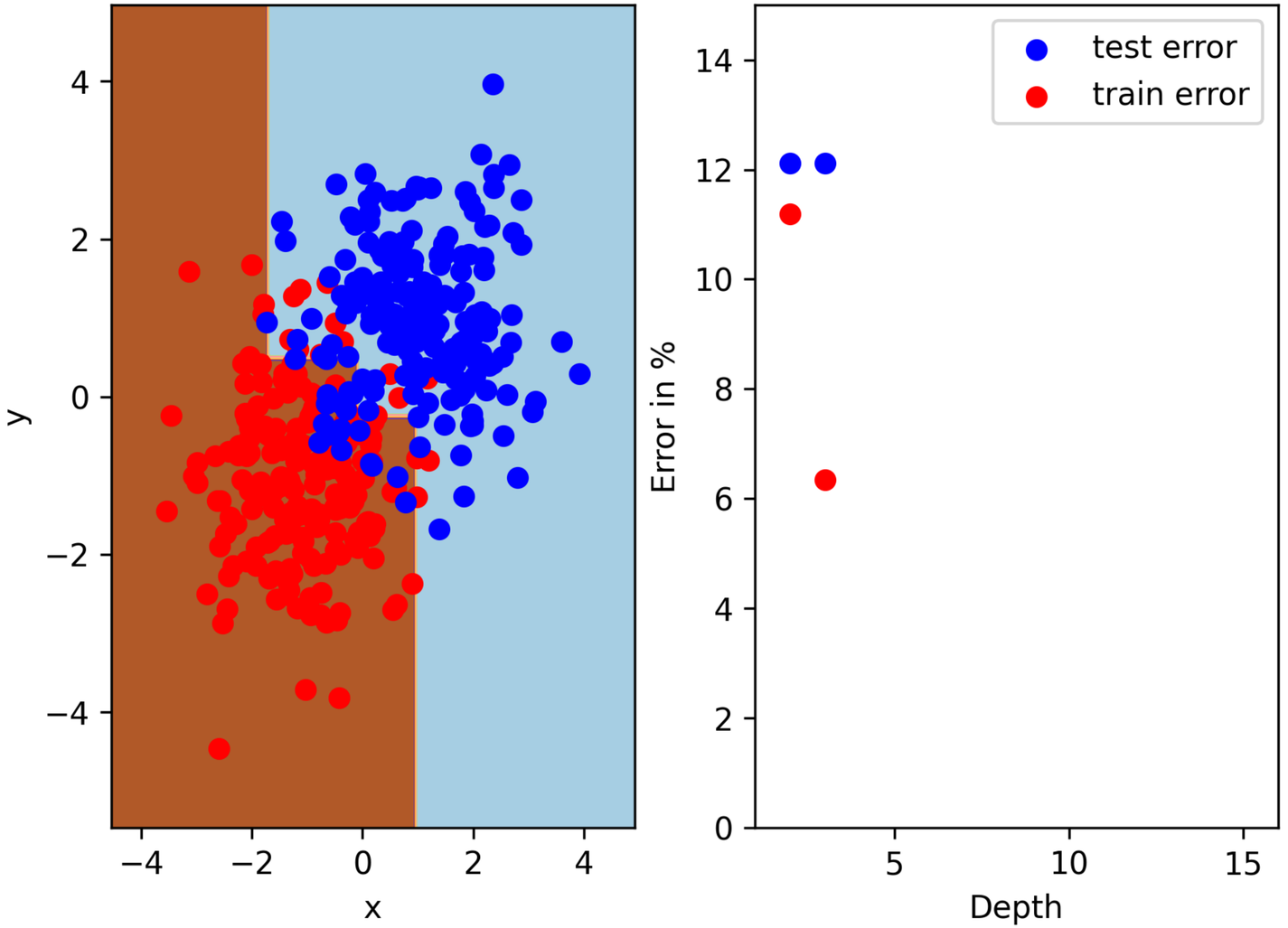
Example

- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)



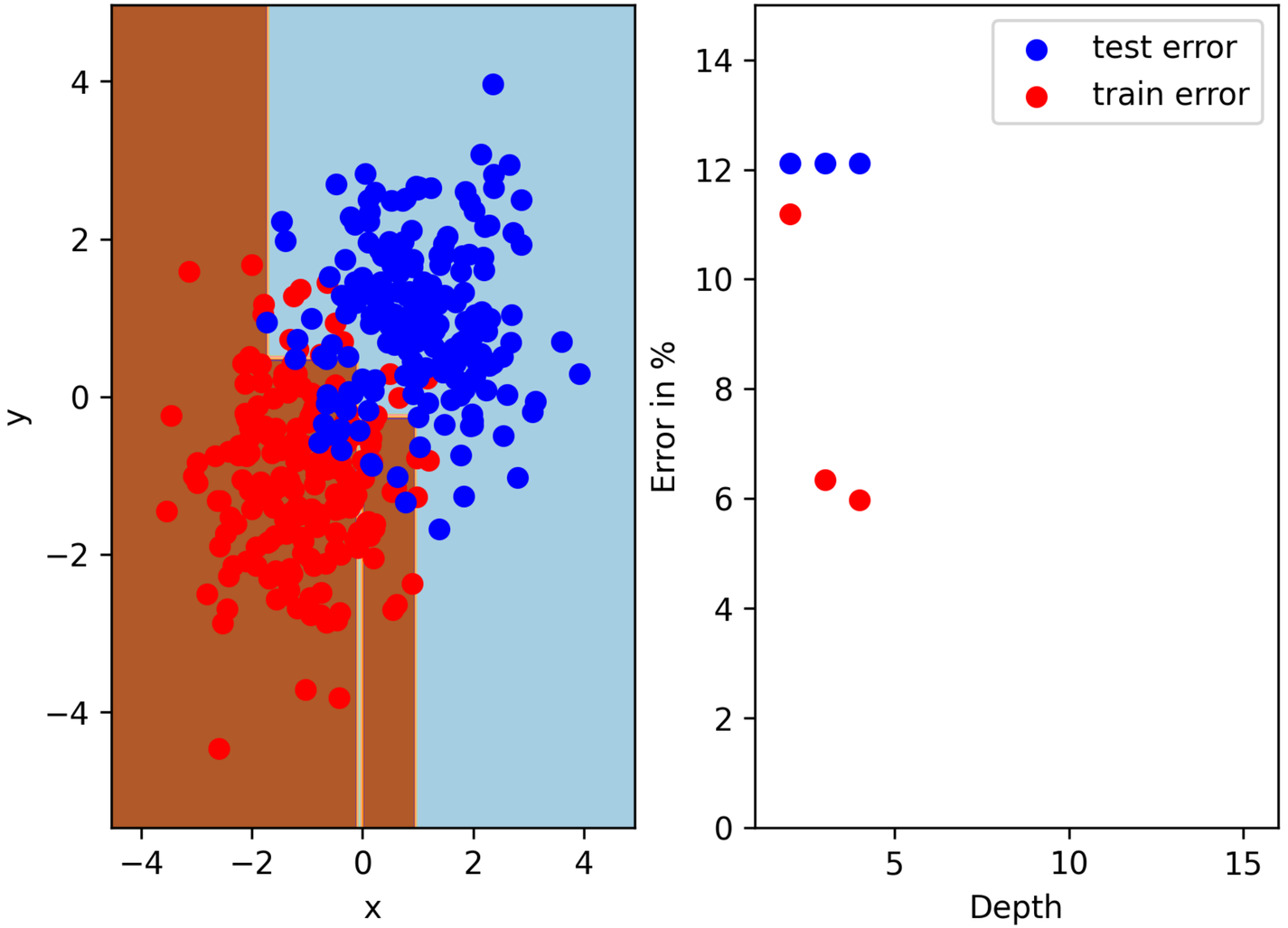
Example

- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)



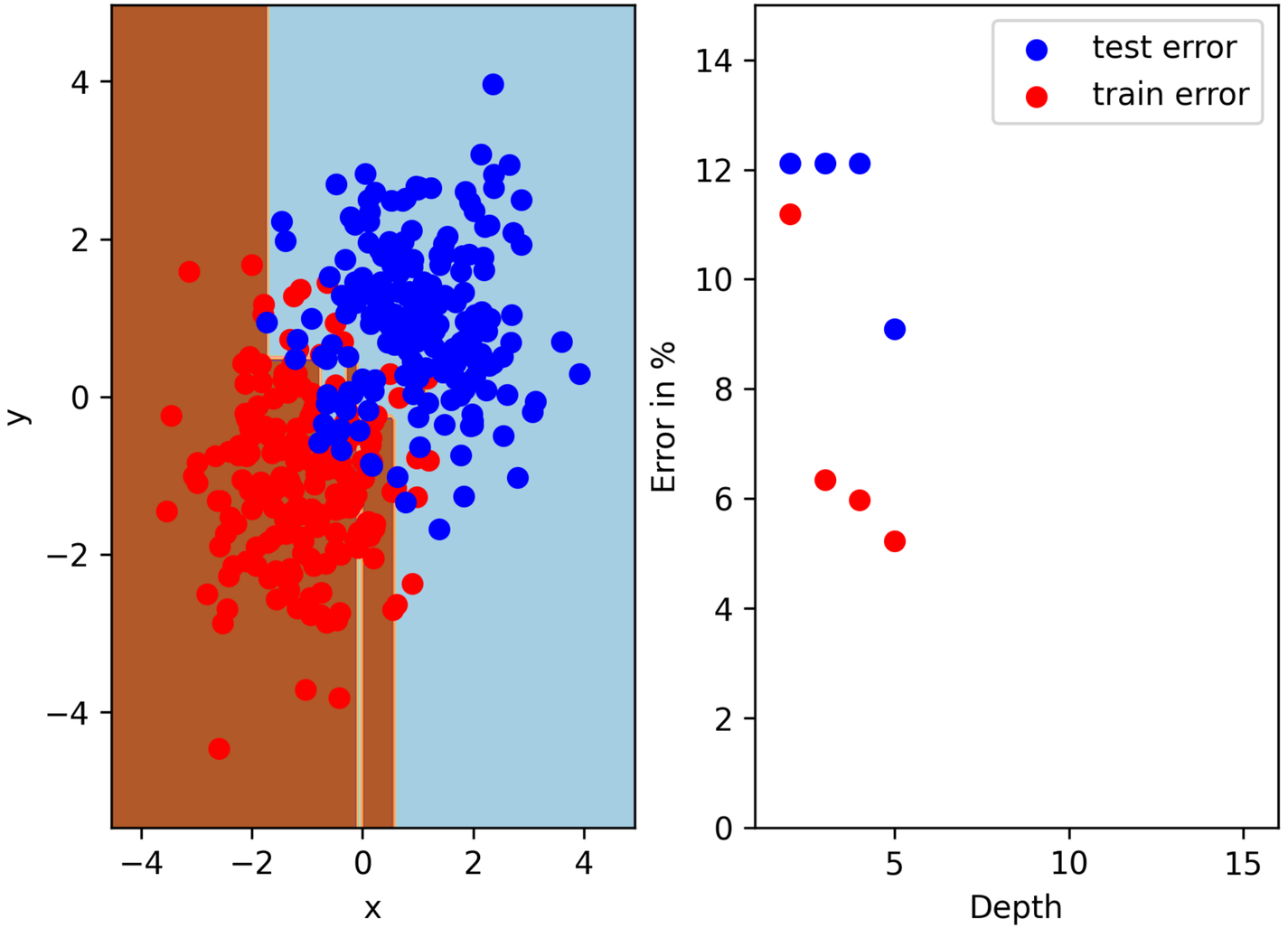
Example

- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)



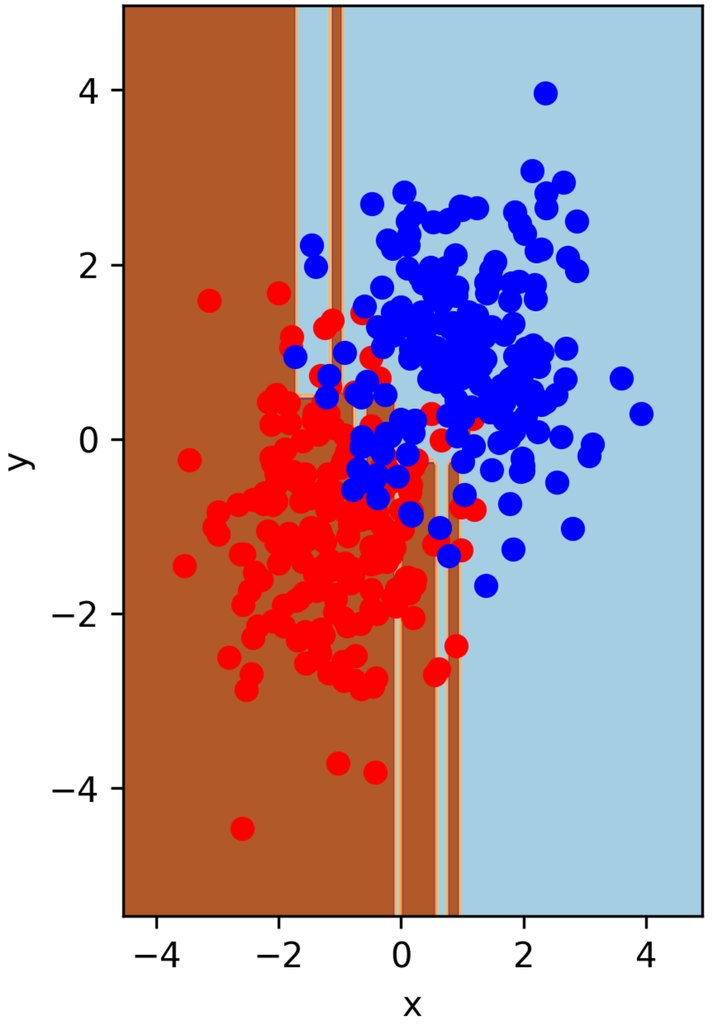
Example

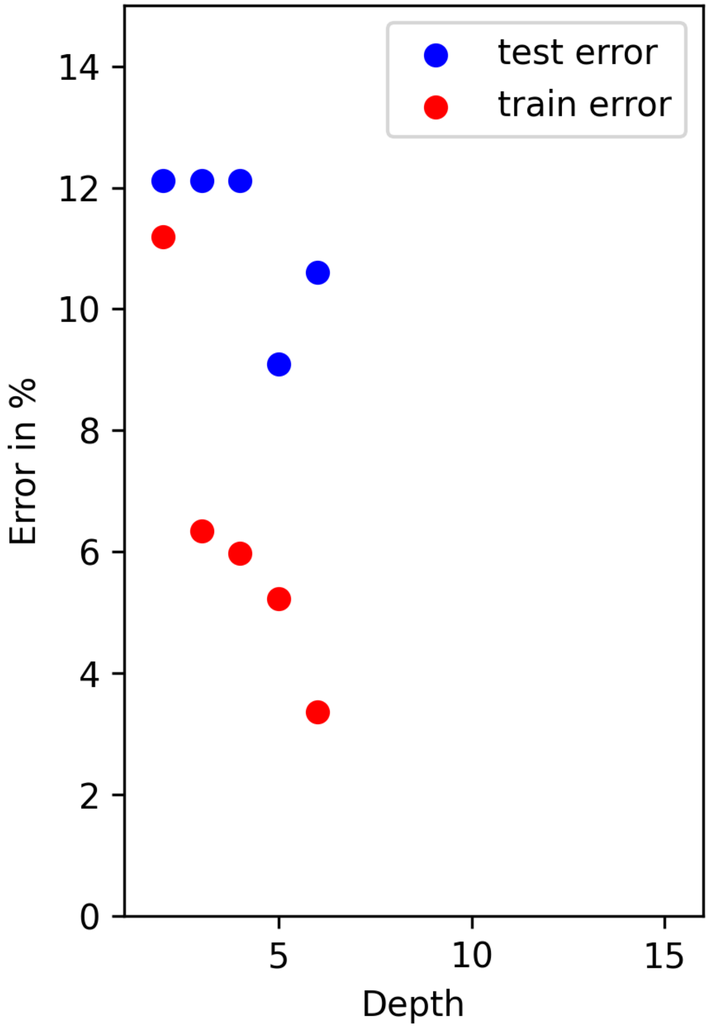
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)



Example

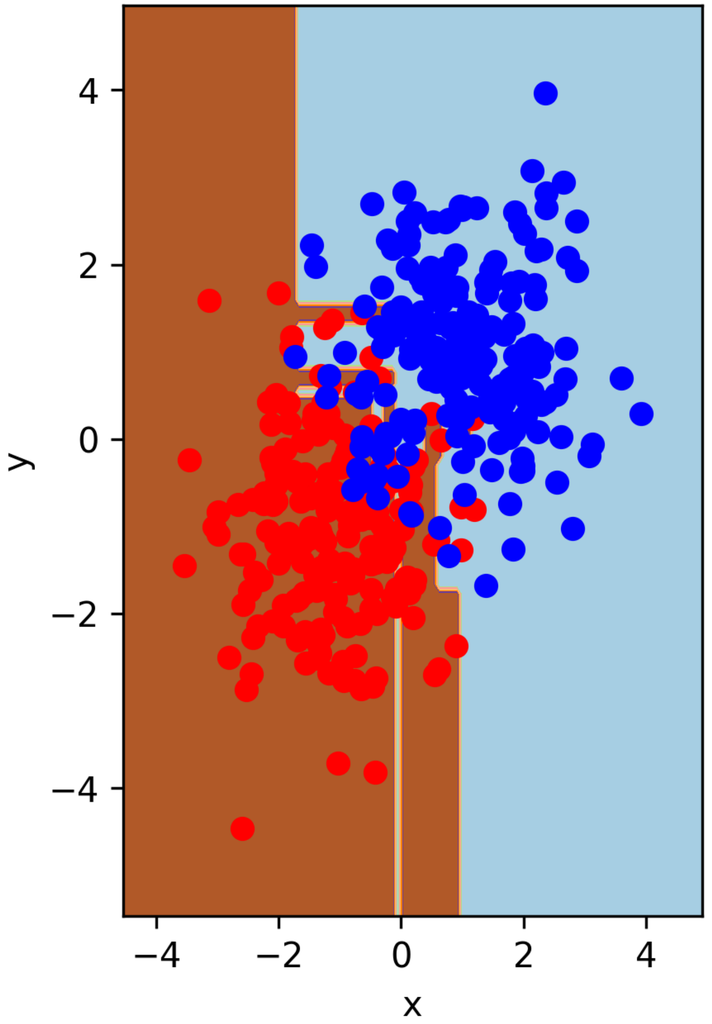
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

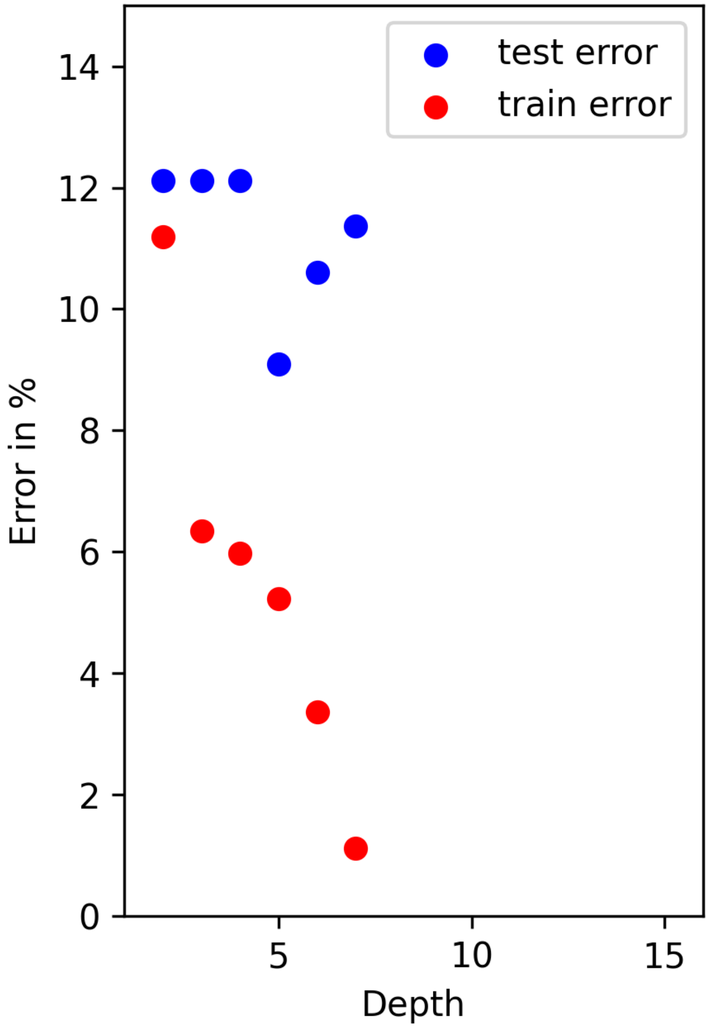




Example

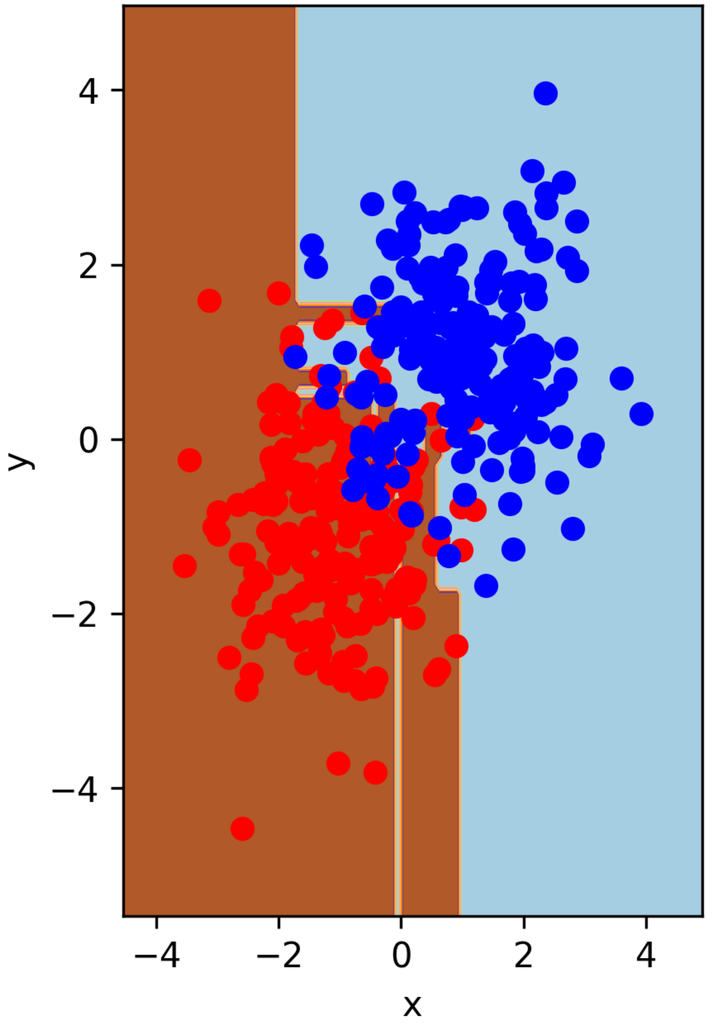
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

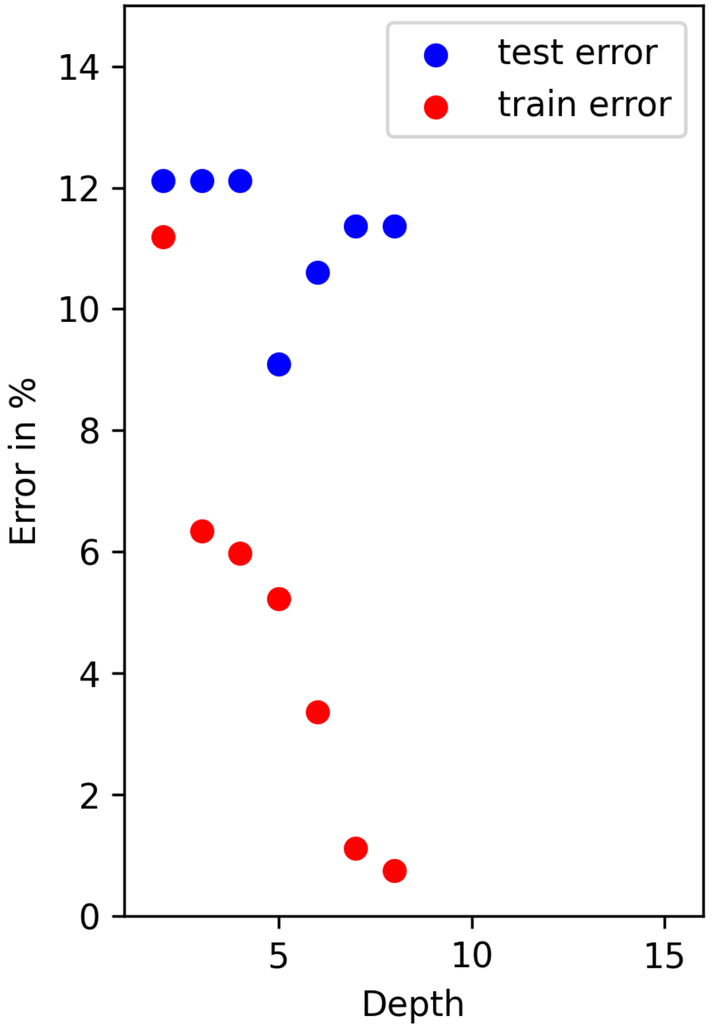




Example

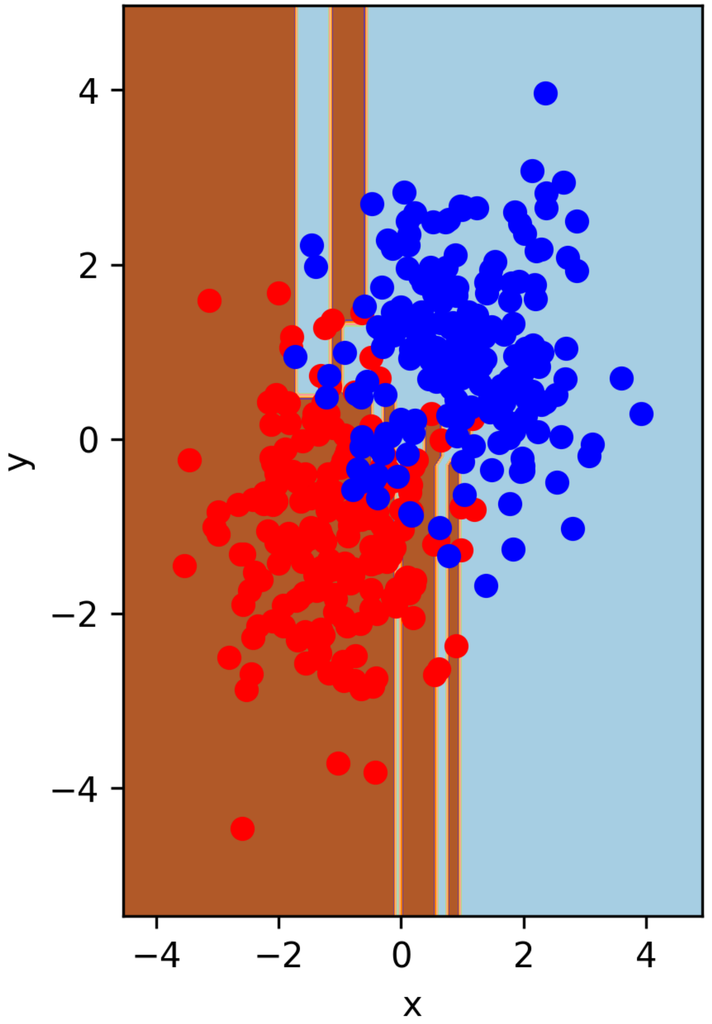
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

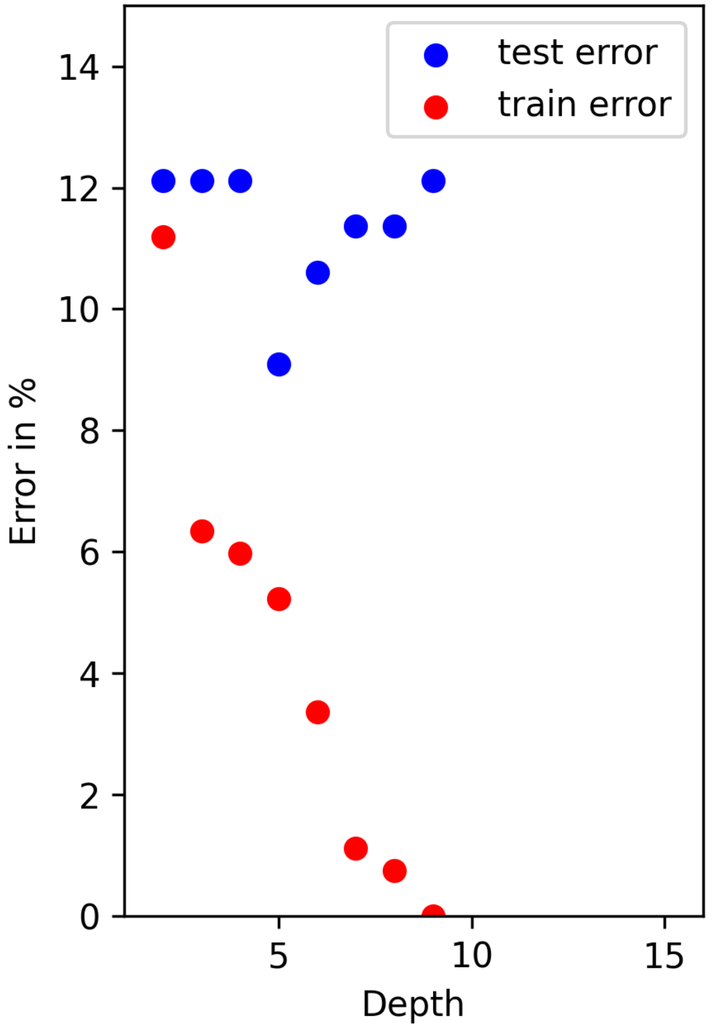




Example

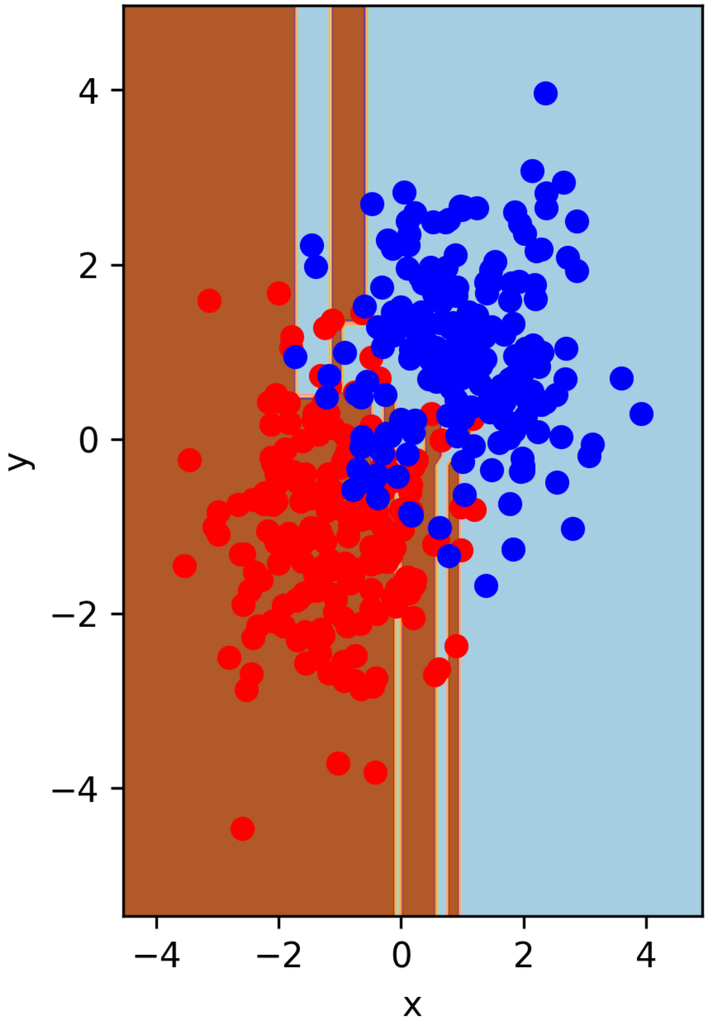
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

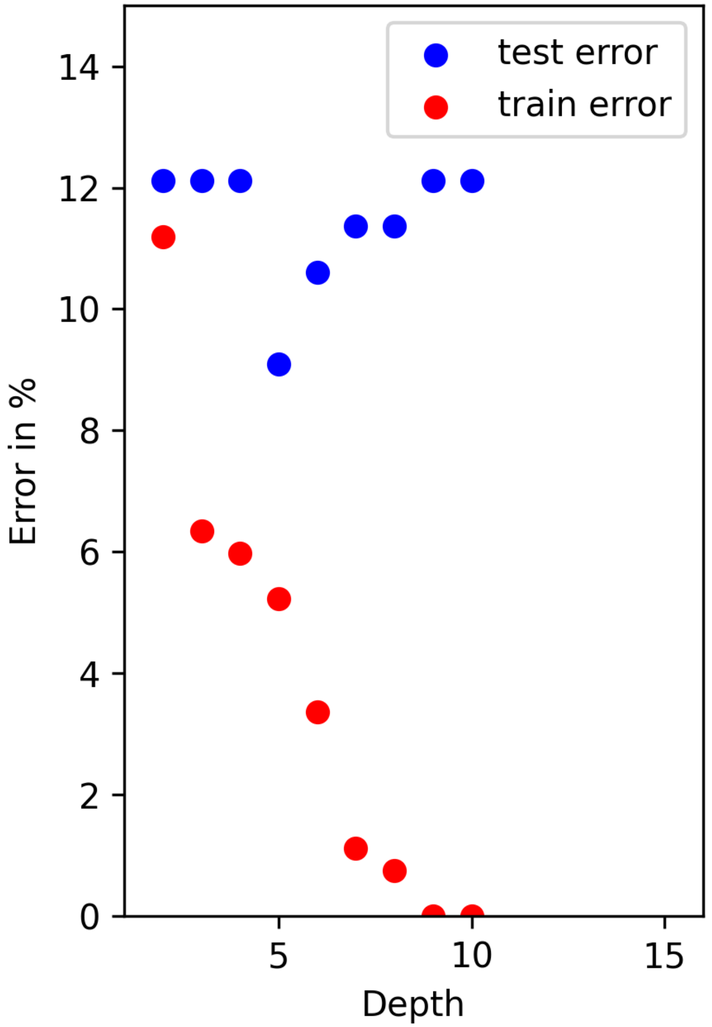




Example

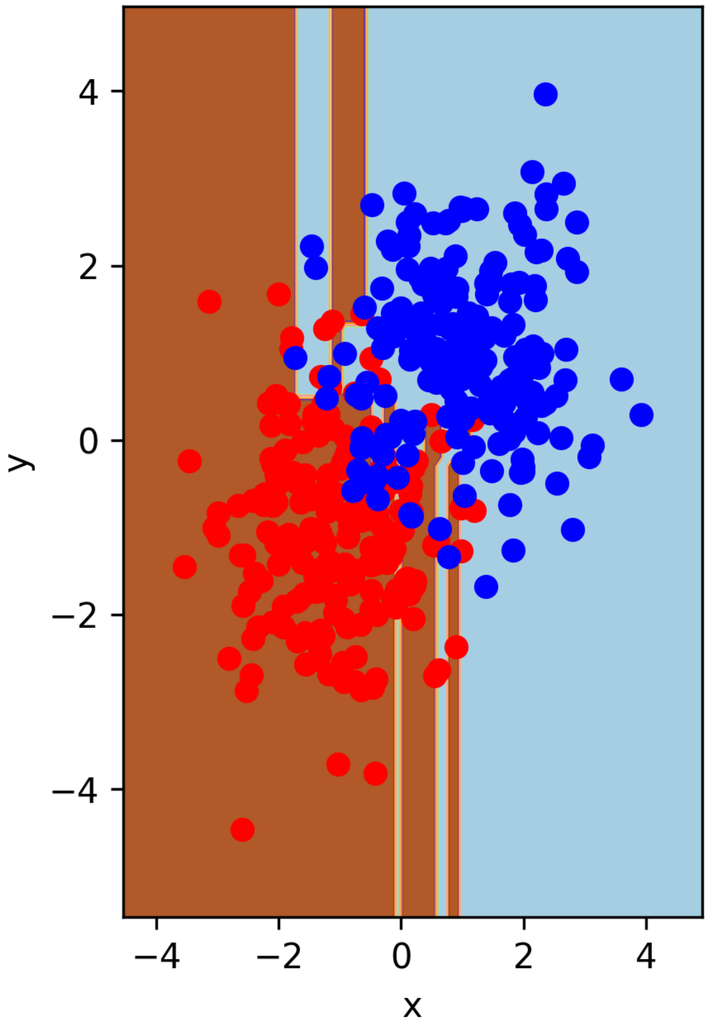
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

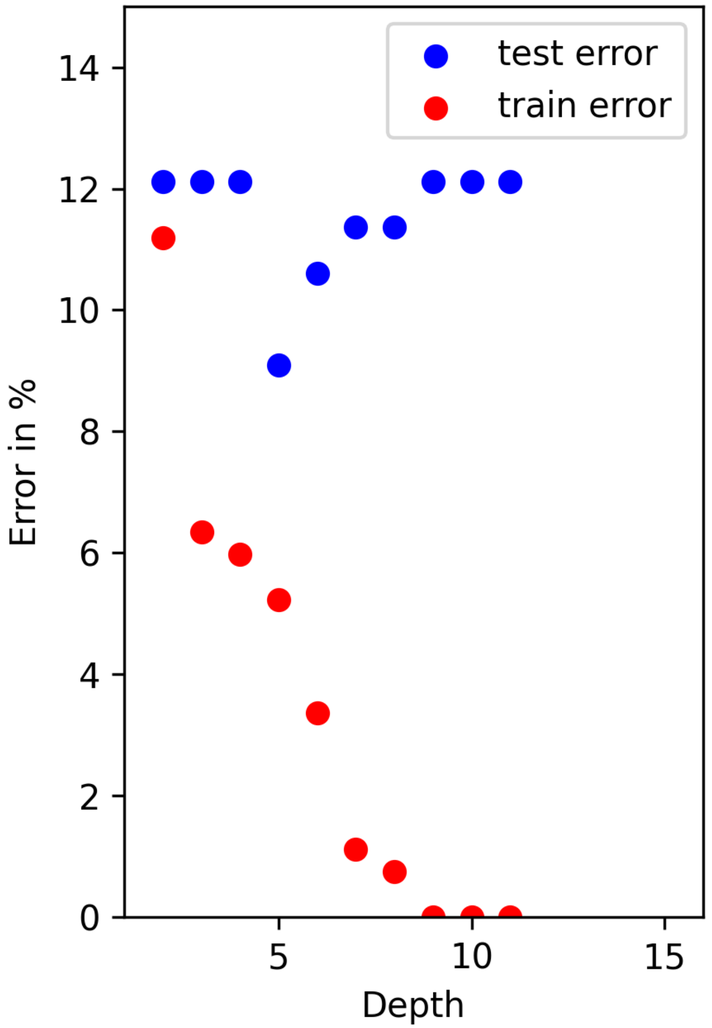




Example

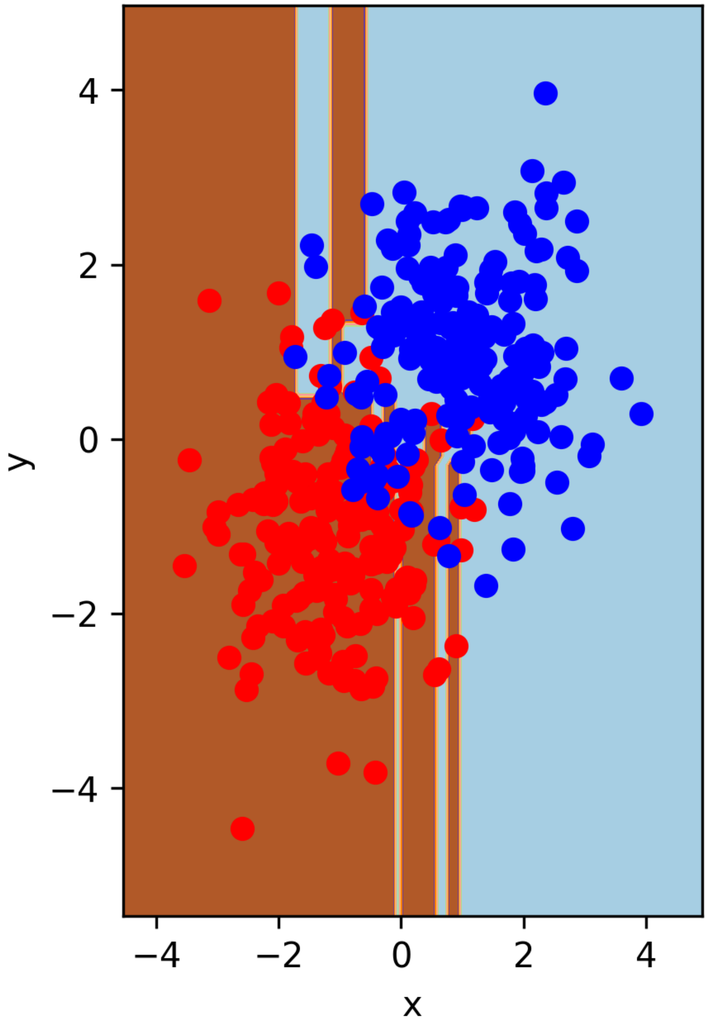
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

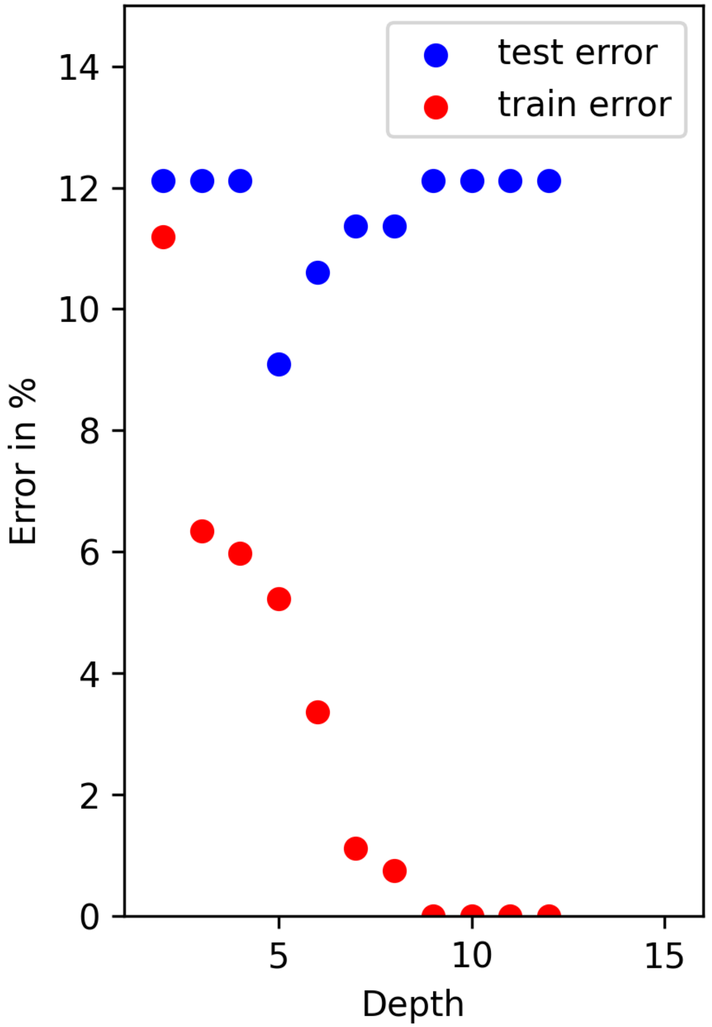




Example

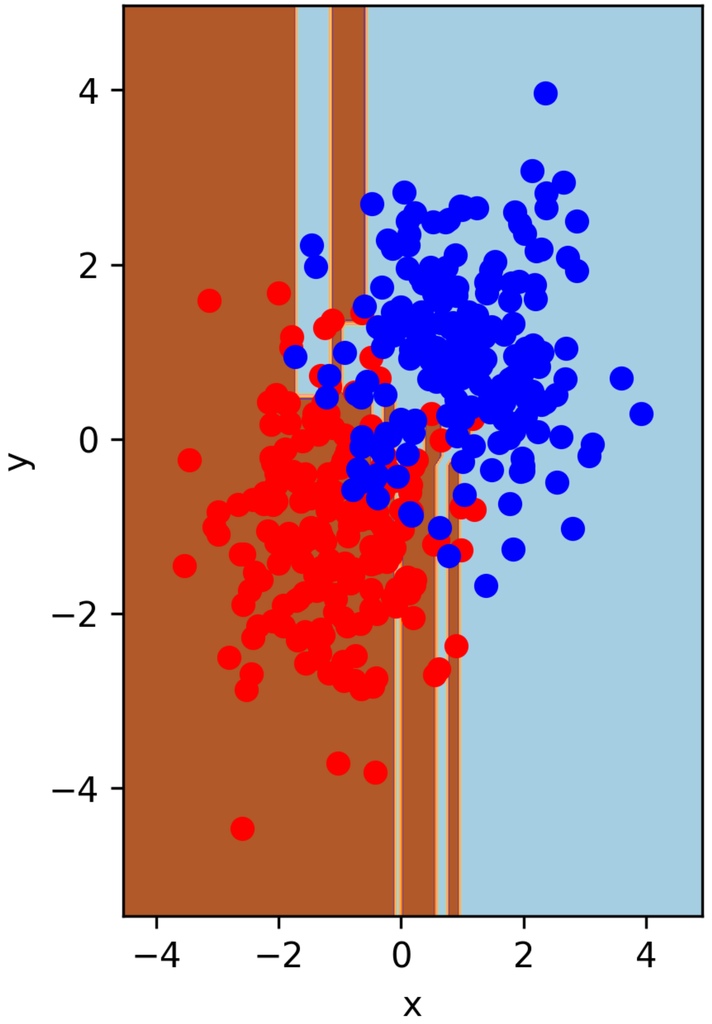
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

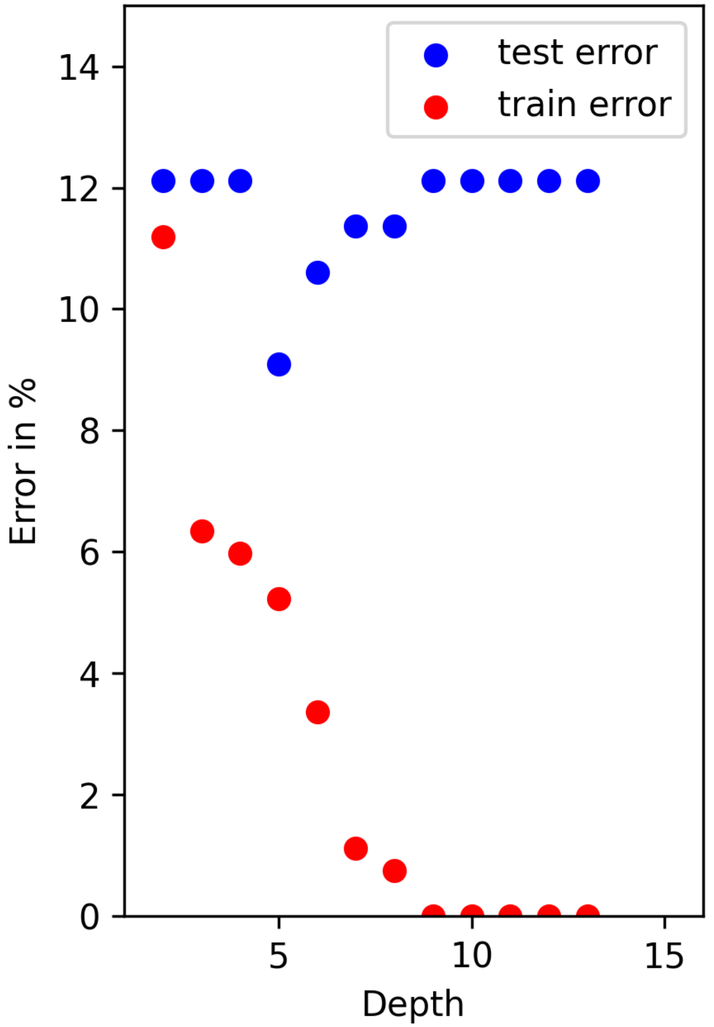




Example

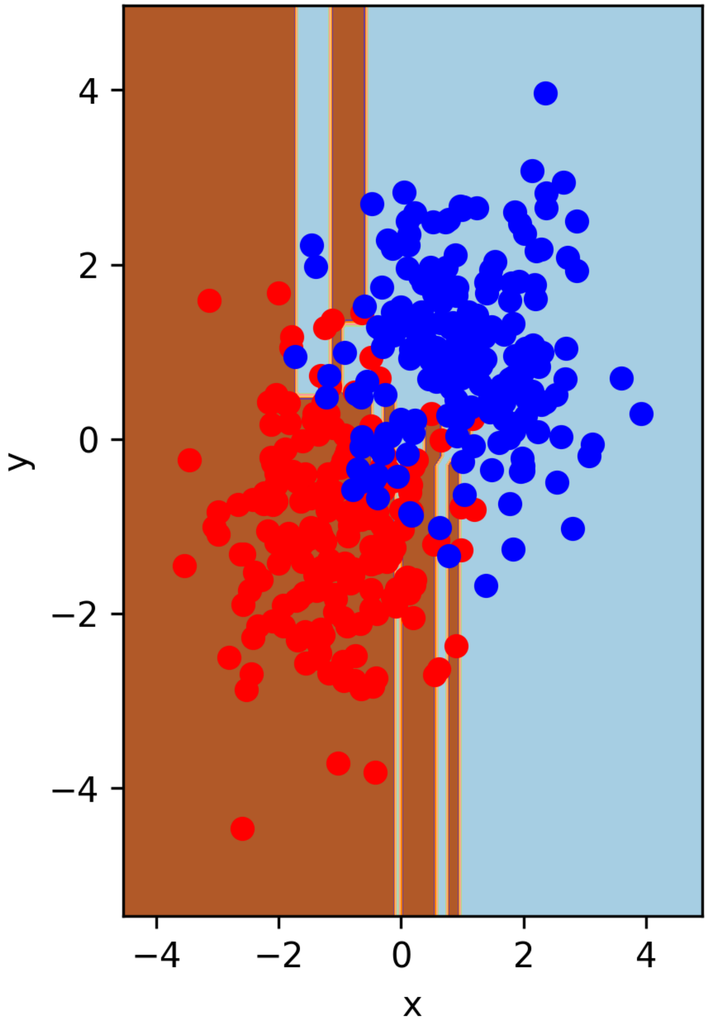
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

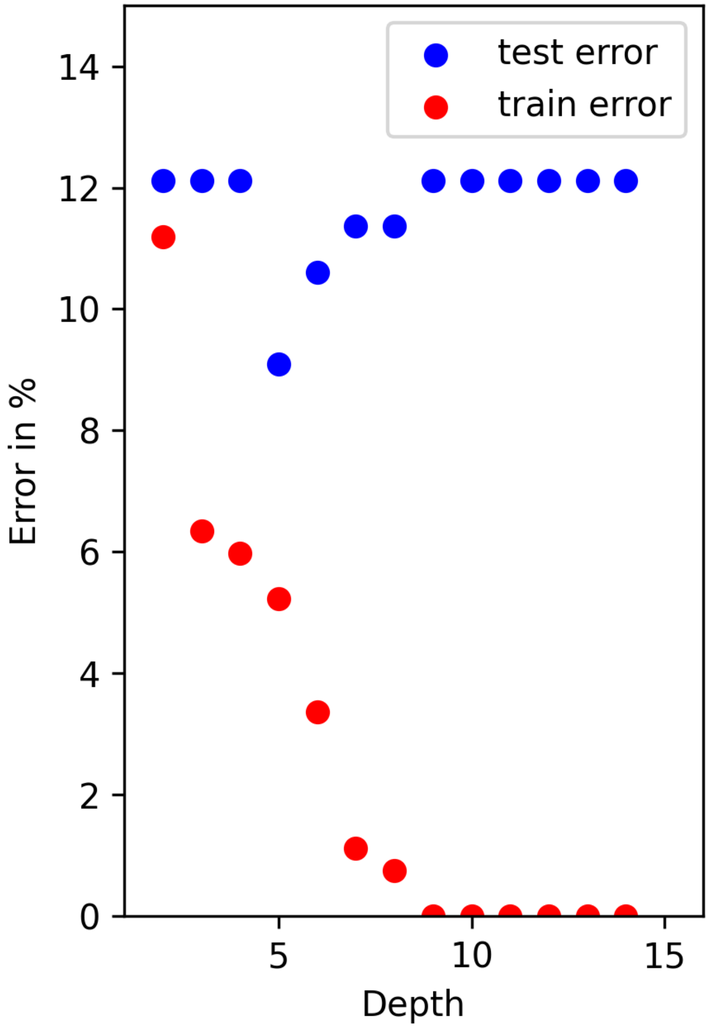




Example

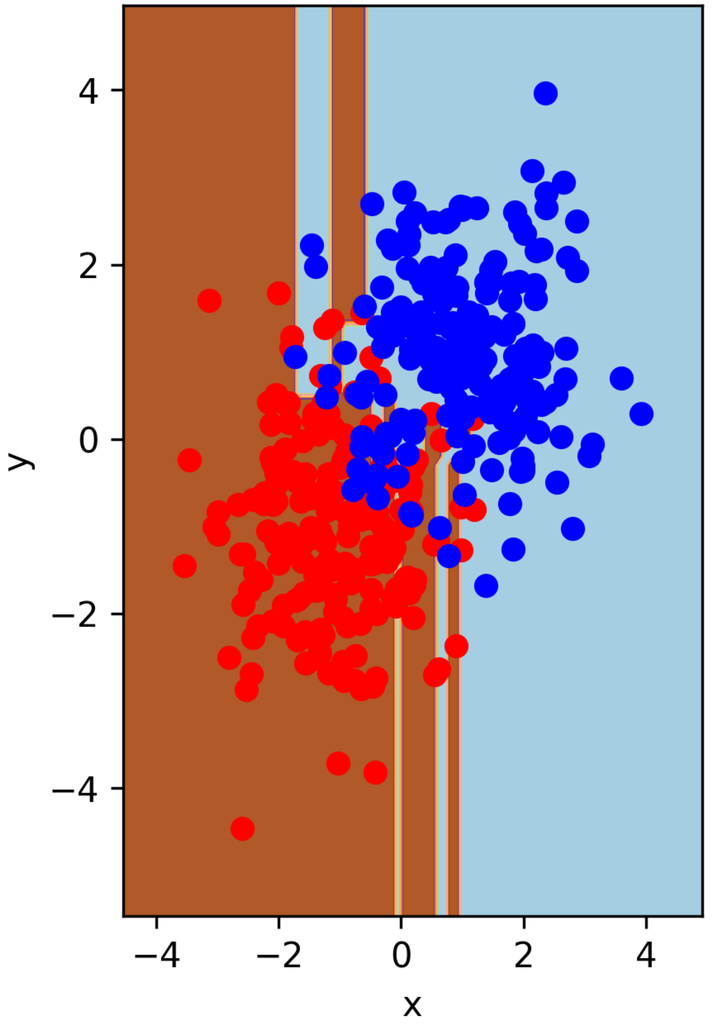
- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)

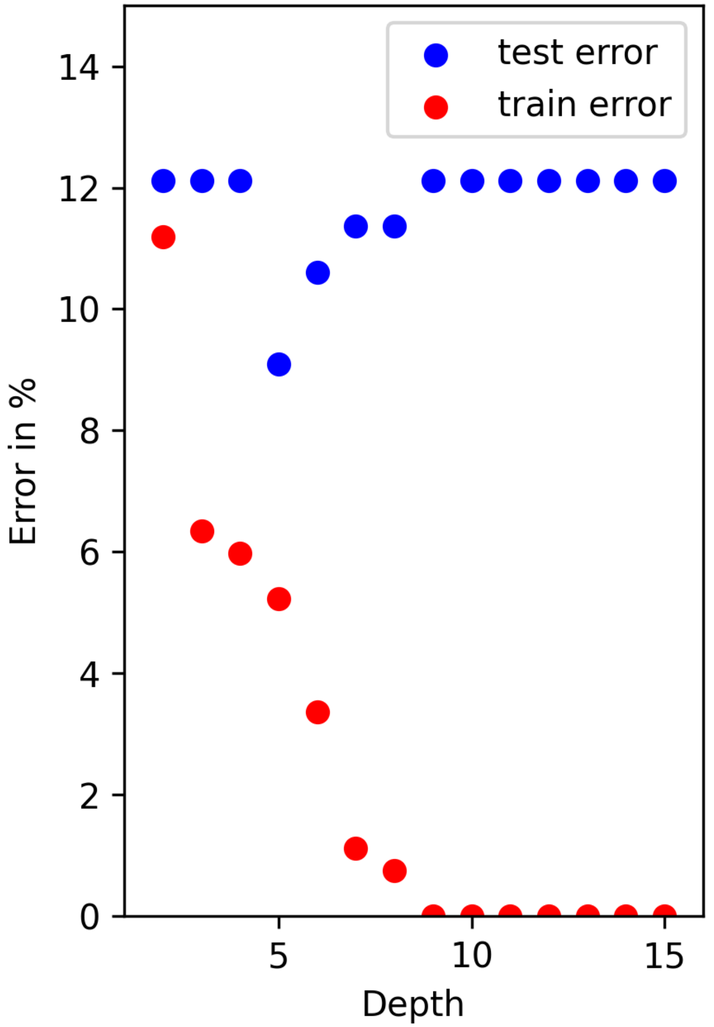




Example

- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- At low depth test error approx equal to train error i.e. we have a bias problem
- As depth increases train error goes to zero but training error stays constant we have a variance problem! (it doesn't generalise!)





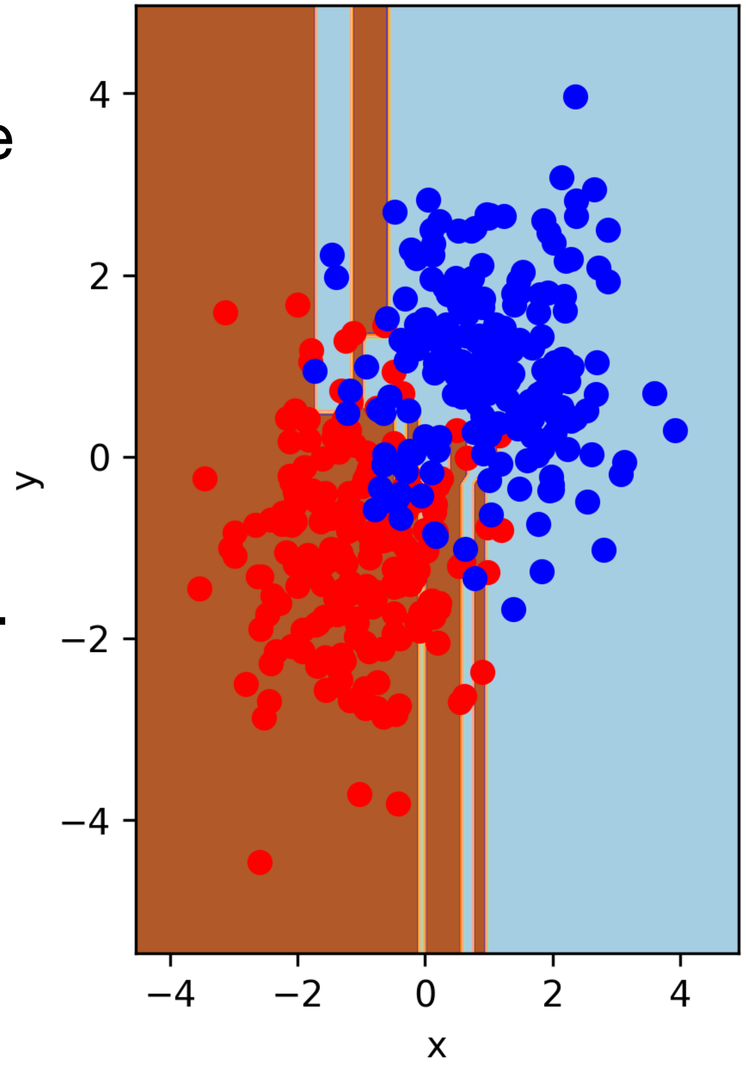
Decision Trees and Bias/Variance Problems

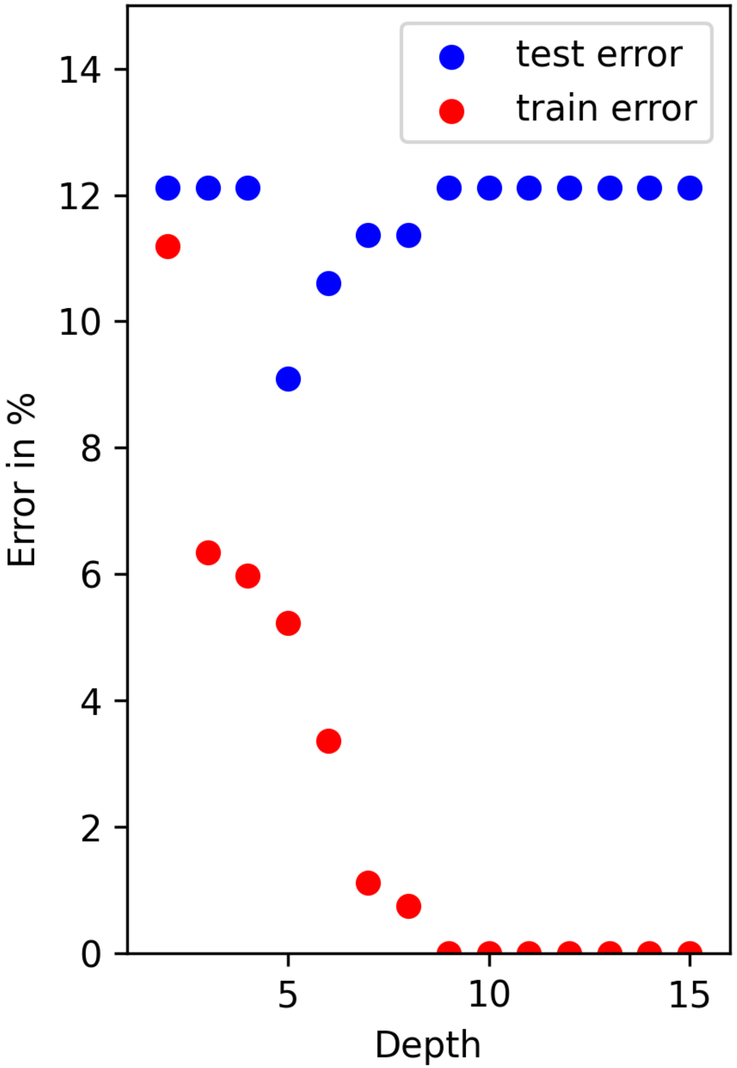
In other classification algorithms we “regularise” using a real number

In decision trees, we can only vary the depth (integer and not fine-grained enough)

Bagging takes a different approach. We build deep trees and get a variance problem.

We use **Bagging** (Random Forests) to address the variance problem.





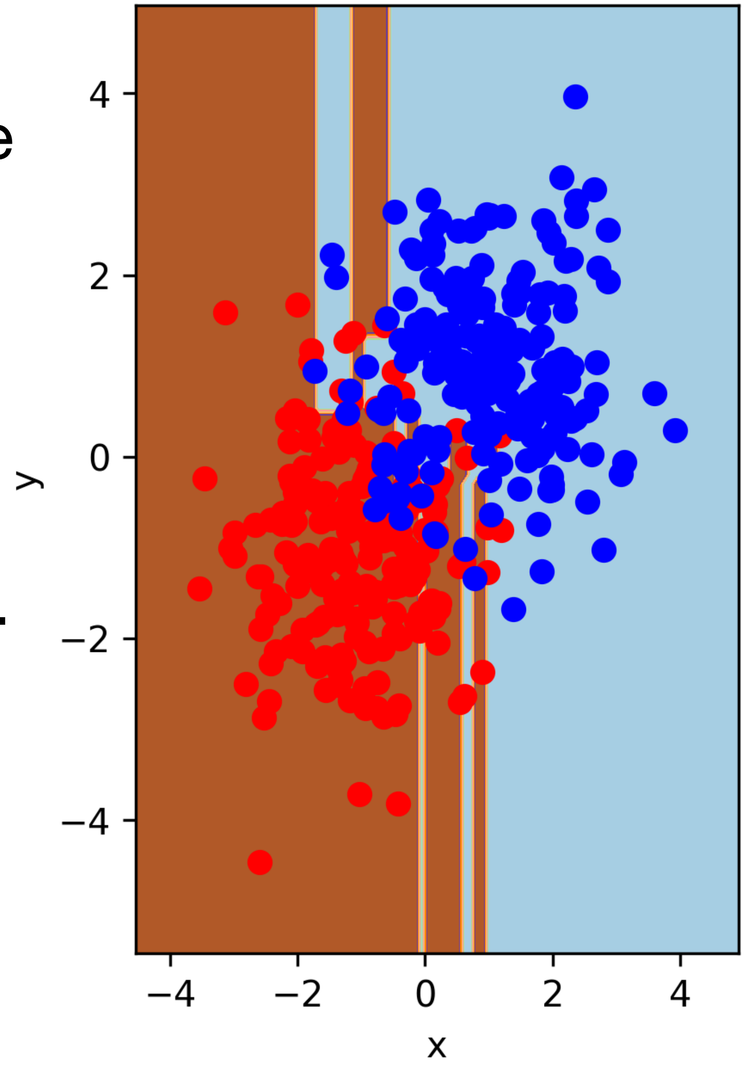
Decision Trees and Bias/Variance Problems

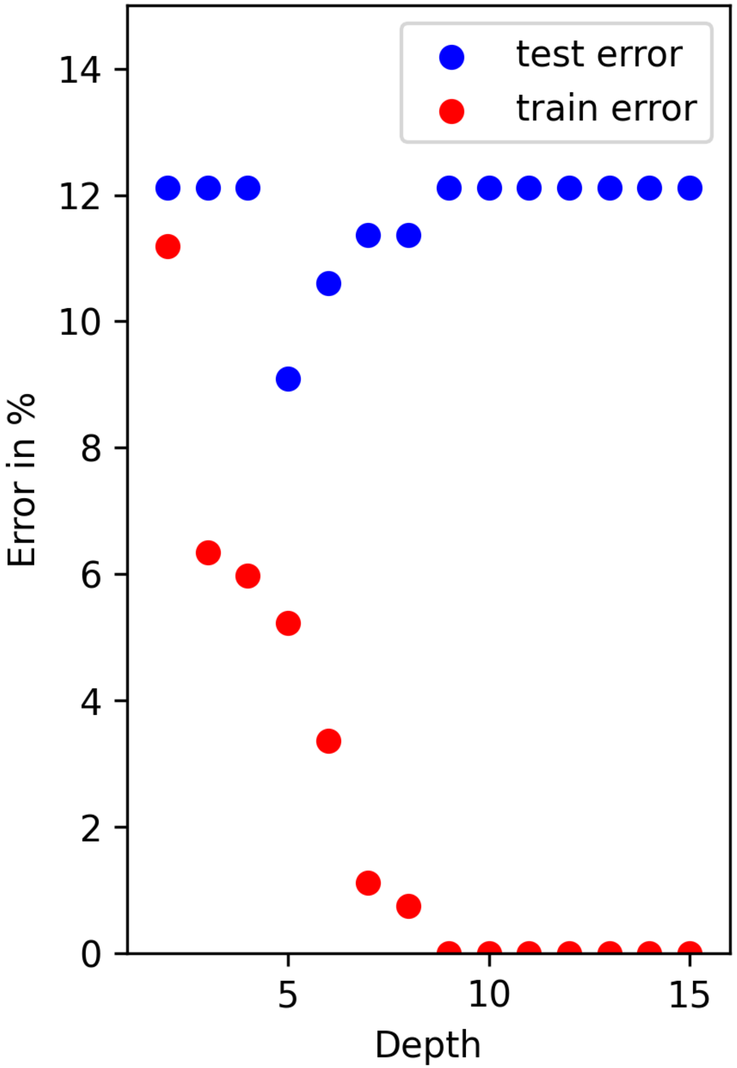
In other classification algorithms we “regularise” using a real number

In decision trees, we can only vary the depth (integer and not fine-grained enough)

Bagging takes a different approach. We build deep trees and get a variance problem.

We use **Bagging** (Random Forests) to address the variance problem.





Reducing Variance — The ideal case

$$\mathbb{E}_{\mathscr{D}}\left[\left(f(x;\mathscr{D}) - \mathbb{E}[y\,|\,x]\right)^2\right] = \underbrace{\mathbb{E}_{\mathscr{D}}\left[\left(f(x;\mathscr{D}) - \mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})\right)^2\right]}_{\text{Variance}} + \underbrace{\left(\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})] - \mathbb{E}[y\,|\,x]\right)^2}_{\text{Bias}^2}$$

$\mathbb{E}[y\,|\,x]$: ideal predictor $f(x;\mathscr{D})$ classifier trained with data $\mathscr{D}$

Reducing Variance — The ideal case

$$\mathbb{E}_{\mathscr{D}}\left[\left(f(x;\mathscr{D})-\mathbb{E}[y\,|\,x]\right)^2\right]=\underbrace{\mathbb{E}_{\mathscr{D}}\left[\left(f(x;\mathscr{D})-\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})\right)^2\right]}_{\text{Variance}}+\underbrace{\left(\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})]-\mathbb{E}[y\,|\,x]\right)^2}_{\text{Bias}^2}$$

$\mathbb{E}[y\,|\,x]$: ideal predictor $f(x;\mathscr{D})$ classifier trained with data $\mathscr{D}$

Get M more data sets: $\mathscr{D}_1,\ldots,\mathscr{D}_M$

Train M trees with high depth: $f(x,\mathscr{D}_1),\ldots,f(x,\mathscr{D}_M)$

Each tree will have high variance (overfitting)

Reducing Variance — The ideal case

$$\mathbb{E}_{\mathscr{D}}\left[\left(f(x;\mathscr{D})-\mathbb{E}[y\,|\,x]\right)^2\right]=\underbrace{\mathbb{E}_{\mathscr{D}}\left[\left(f(x;\mathscr{D})-\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})\right)^2\right]}_{\text{Variance}}+\underbrace{\left(\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})]-\mathbb{E}[y\,|\,x]\right)^2}_{\text{Bias}^2}$$

$\mathbb{E}[y\,|\,x]$: ideal predictor $f(x;\mathscr{D})$ classifier trained with data $\mathscr{D}$

Get M more data sets: $\mathscr{D}_1,\ldots,\mathscr{D}_M$

Train M trees with high depth: $f(x,\mathscr{D}_1),\ldots,f(x,\mathscr{D}_M)$

Each tree will have high variance (overfitting)

Take average (ensemble) instead: $\hat{f}(x)=\frac{1}{M}\sum_{i=1}^{M}f(x;\mathscr{D}_i)$

By the law of large numbers: $\hat{f}(x)\rightarrow\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})]$ i.e. Variance $\rightarrow 0$ with $M\rightarrow\infty$

Reducing Variance — The ideal case

$$\mathbb{E}_{\mathscr{D}}\left[\left(f(x;\mathscr{D})-\mathbb{E}[y\,|\,x]\right)^2\right]=\underbrace{\mathbb{E}_{\mathscr{D}}\left[\left(f(x;\mathscr{D})-\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})\right)^2\right]}_{\text{Variance}}+\underbrace{\left(\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})]-\mathbb{E}[y\,|\,x]\right)^2}_{\text{Bias}^2}$$

$\mathbb{E}[y\,|\,x]$: ideal predictor $f(x;\mathscr{D})$ classifier trained with data $\mathscr{D}$

Get M more data sets: $\mathscr{D}_1,\ldots,\mathscr{D}_M$

Train M trees with high depth: $f(x,\mathscr{D}_1),\ldots,f(x,\mathscr{D}_M)$

Each tree will have high variance (overfitting)

Take average (ensemble) instead: $\hat{f}(x)=\frac{1}{M}\sum_{i=1}^{M}f(x;\mathscr{D}_i)$

By the law of large numbers: $\hat{f}(x)\rightarrow\mathbb{E}_{\mathscr{D}}[f(x;\mathscr{D})]$ i.e. Variance $\rightarrow 0$ with $M\rightarrow\infty$

Whats the catch? We can't generate M i.i.d. data sets!

Bagging (Bootstrap + Aggregating)

Instead of getting M i.i.d more data sets, we sample with replacement

$$\mathcal{D}_1, \ldots, \mathcal{D}_M$$

Train M trees with high depth: $f(x, \mathcal{D}_1), \ldots, f(x, \mathcal{D}_M)$

Each tree will have high variance (overfitting)

Take average (ensemble) instead: $\hat{f}(x) = \frac{1}{M} \sum_{i=1}^{M} f(x; \mathcal{D}_i)$

Law of large numbers no longer applies!

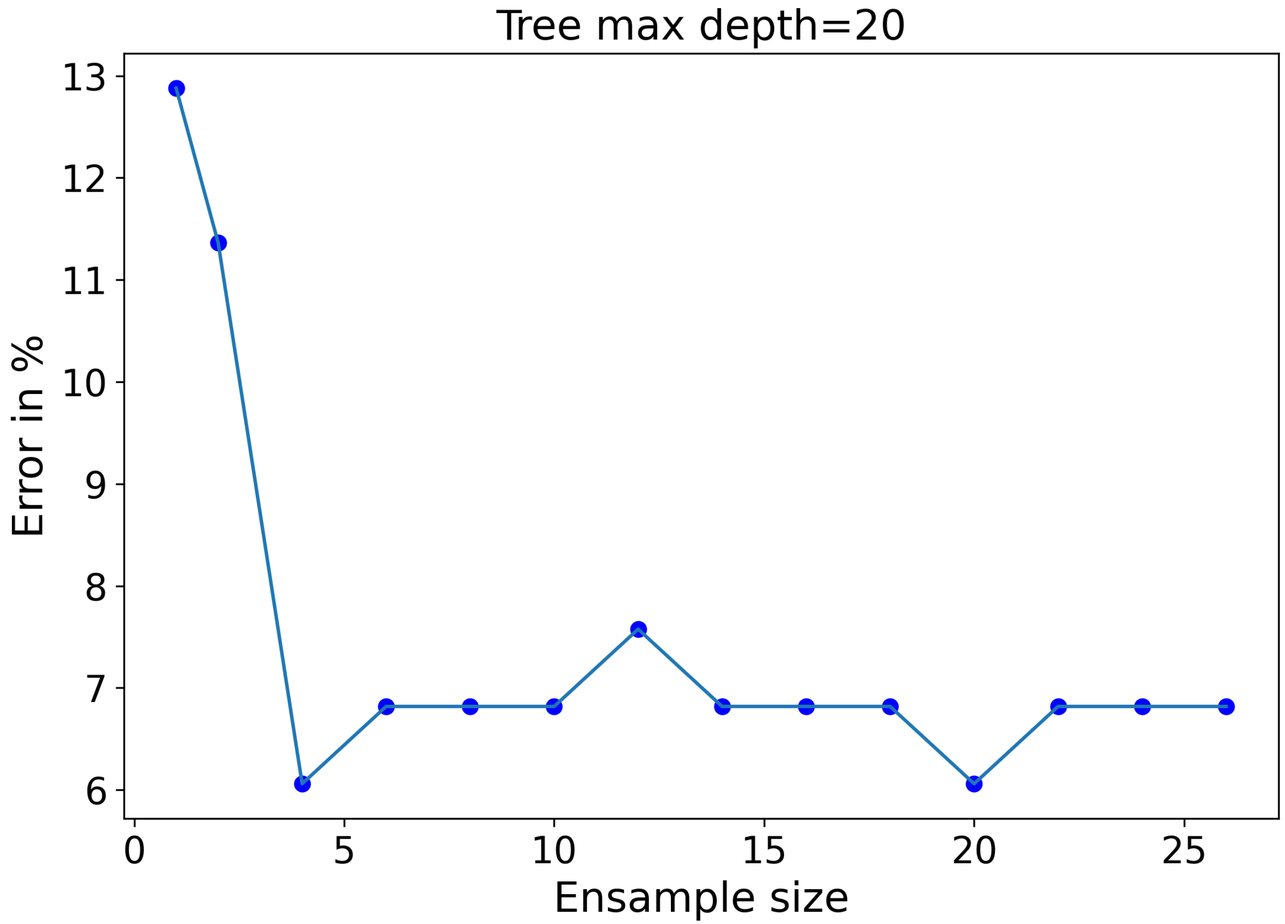
Each new data set has the same size as the original

All M new data sets have the same distribution as the original but are not independent

In practice it still works, and is very easy to try!

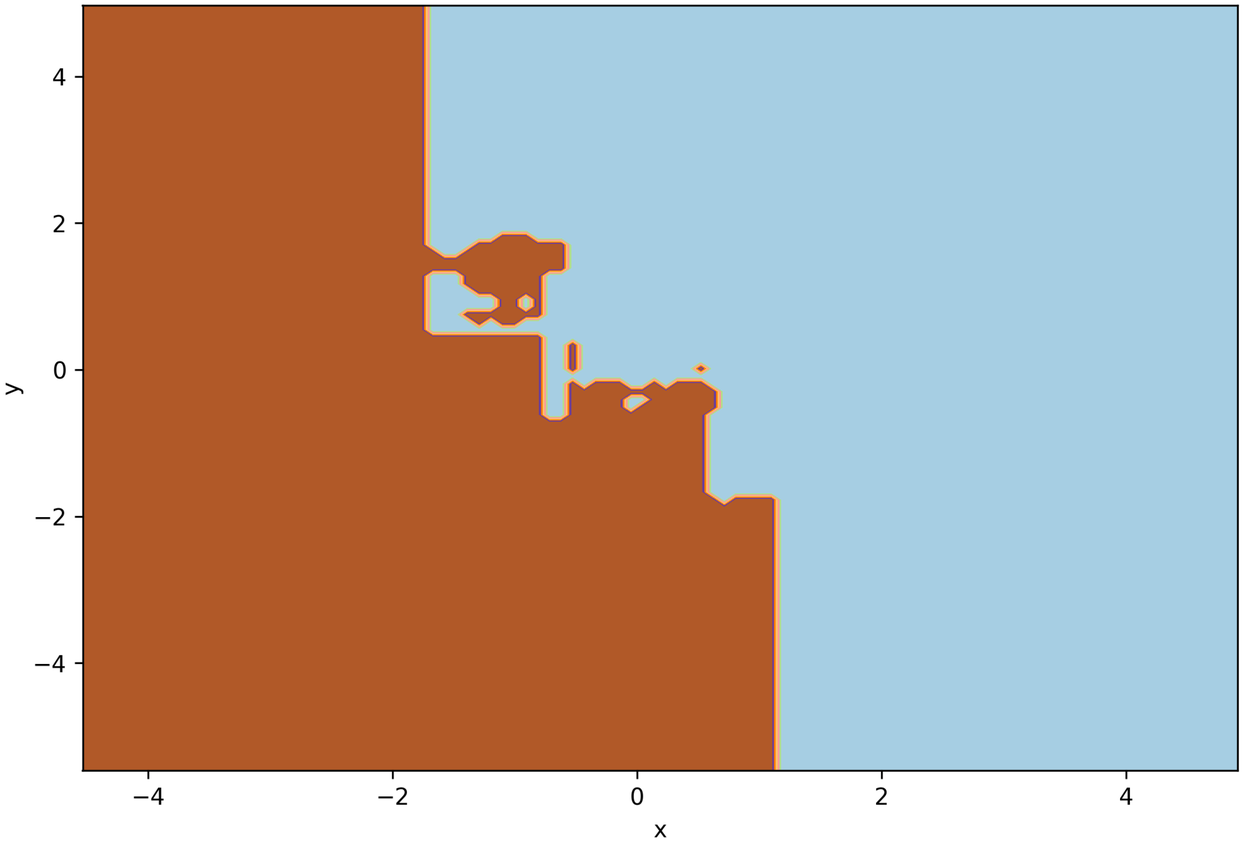
Bagging Demo

- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- Single tree gives error 13% approx
- Reduce the error by 50% just by using an ensemble of trees.

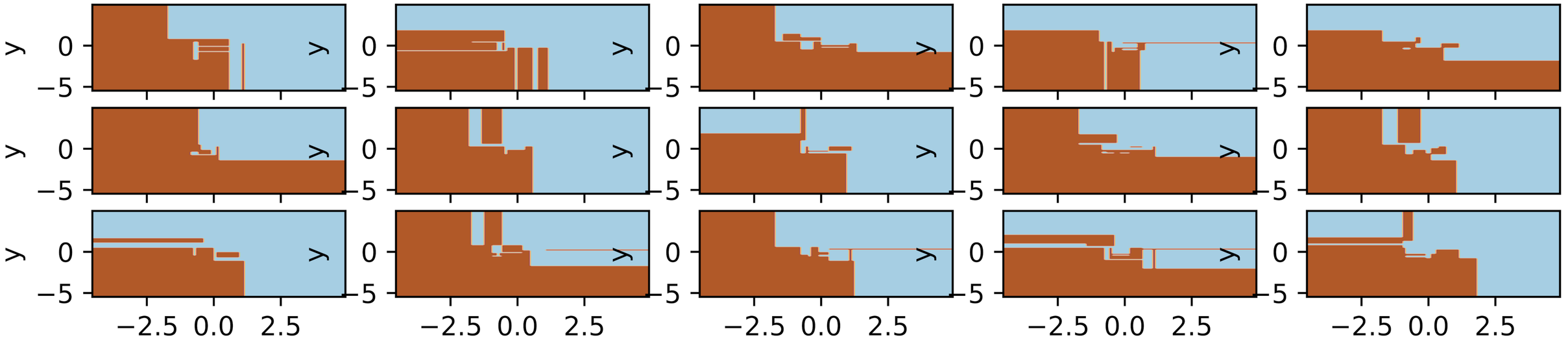


Bagging Demo

- Fit two Gaussian distributions centred at (1,1) and (-1,-1)
- Single tree gives error 13% approx
- Reduce the error by 50% just by using an ensemble of trees.



Bagging Demo



Random Forests

The **Random Forest** is a powerful bagging algorithm:

Sample M data sets from the original dataset, $\mathscr{D}$: $\mathscr{D}_1, \ldots, \mathscr{D}_M$

Train M trees to maximum depth: $f(x, \mathscr{D}_1), \ldots, f(x, \mathscr{D}_M)$

Randomly sub-sample (without replacement) k features before each split

Sub-sampling *decorrelates* the trees and tends to improve generalisation

A good rule of thumb is $k = \sqrt{d}$ (where d are the original number of features)

Final estimator is averaged $\hat{f}(x) = \frac{1}{M} \sum_{i=1}^{M} f(x; \mathscr{D}_i)$

Summary

- **Classification and Regression Trees** (CART) vanilla implementation tends to have high variance or high bias
- **Ensemble Learning**: Create multiple models (trees) and average them (helps with stability/variance)
- **Stacking**: Like Ensemble Learning but with weighted averages
- **Bagging**: Resample the data so that the tree is trained on slightly different data
- **Random Forests**: A bagging algorithm that, in addition, randomly sub-samples with replacement features before each split

Bagging - Theoretical Underpinnings

Bagging, ensemble learning and others build models of the form:

$$f(x;\theta) = \sum_{m=1}^{M} \beta_m F_m(x;\theta)$$

where F_m is the mth tree, θ are the parameters and β_m is the weight typically $1/M$

The result is an **additive** model, the trees act like basis functions (and could be anything, even a NN!)

The goal is to minimise: $\min_f \sum_{i=1}^{N} L(y_i, f(x_i))$

Note that we are minimising over all possible functions which is not practical!

But this point of view is useful for developing new algorithms

Boosting

Additive Models are extremely powerful and boosting finds an approximate solution

Boosting: Trees are grown *sequentially* using information obtained via previous trees. Each tree is typically (low depth). Each tree fits a modified version of the original data set.

Why sequentially? Instead of fitting a single high-depth tree and overfitting, the idea is to learn *iteratively.*

At every iteration instead of trying to find a better prediction we simply try to fix previous mistakes.

Boosting

Boosting: Combine weak learning algorithms to produce a powerful learning algorithm

Weak learning algorithm: An algorithm that is only *slightly* better than chance, e.g. decision tree with low depth (high bias)

Boosting can be applied to many ML algorithms but is very popular with decision trees

Boosting can help if you model has a bias problem

Boosting - Forward Stage-wise Modelling

Instead of solving: $\min_f \sum_{i=1}^{N} L(y_i, f(x_i))$

Perform the following approximation:

$$(\beta_m, \theta_m) = \arg\min_{\beta,\theta} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta F(x_i, \theta))$$

where $F(x_i, \theta)$ is a weak learner and β is a step-size

we then set $f_m(x) = f_{m-1}(x) + \beta_m F(x; \theta)$

This is very general and the details depend on the choice of loss function and F.

Quadratic Loss and Least Squares Boosting

We use the following loss function: $L(y, x) = (y - x)^2$

the ith term in the objective function becomes

$$L(y_i, f_{m-1}(x_i) + \beta F(x_i, \theta)) = (y_i - f_{m-1}(x_i) - \beta F(x_i, \theta))^2 = (r_{im} - \beta F(x_i, \theta))^2$$

where $r_{im} = y_i - f_{m-1}(x_i)$ i.e. the error the previous model made

To solve this we set $\beta = 1$ and fit F

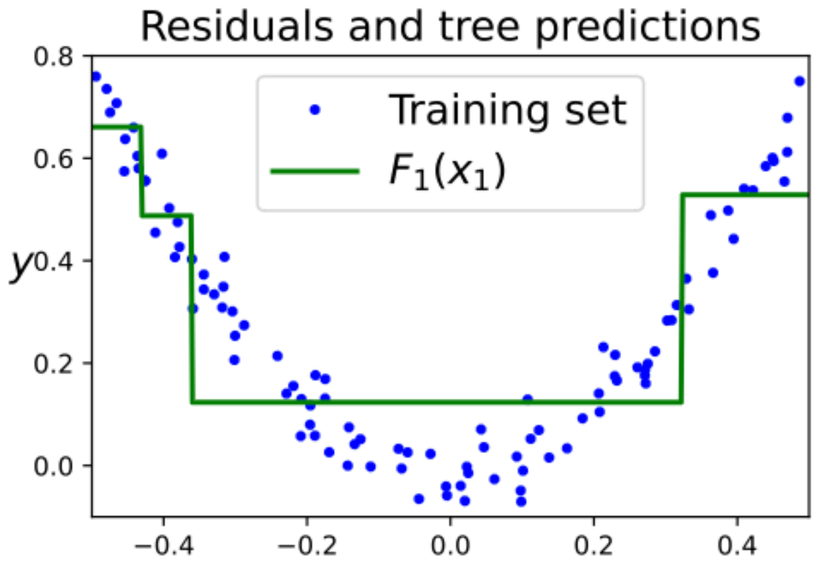
The resulting algorithm is called **least squares boosting**

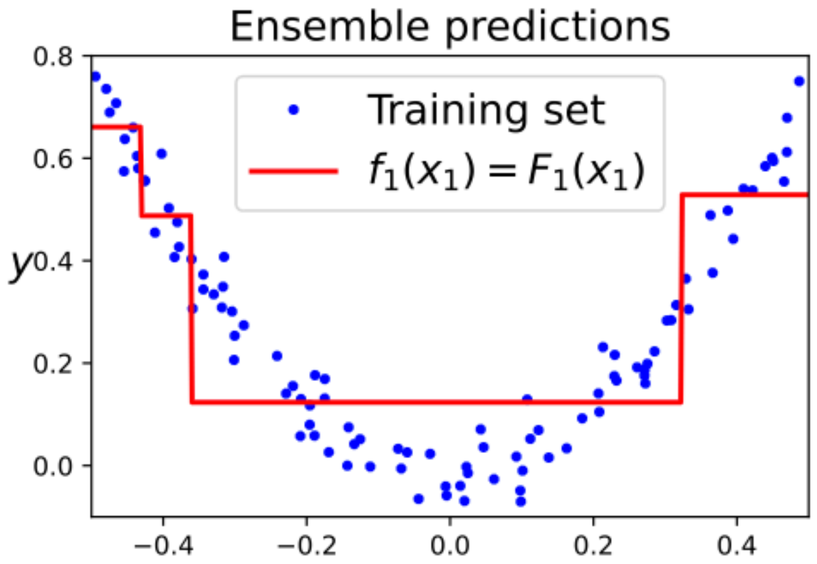
Quadratic Loss and Least Squares Boosting

Training Set:

$y = 3x^2 + 0.05z$, where $z \sim N(0,1)$

$(y_i, x_i), i = 1, \ldots, N$





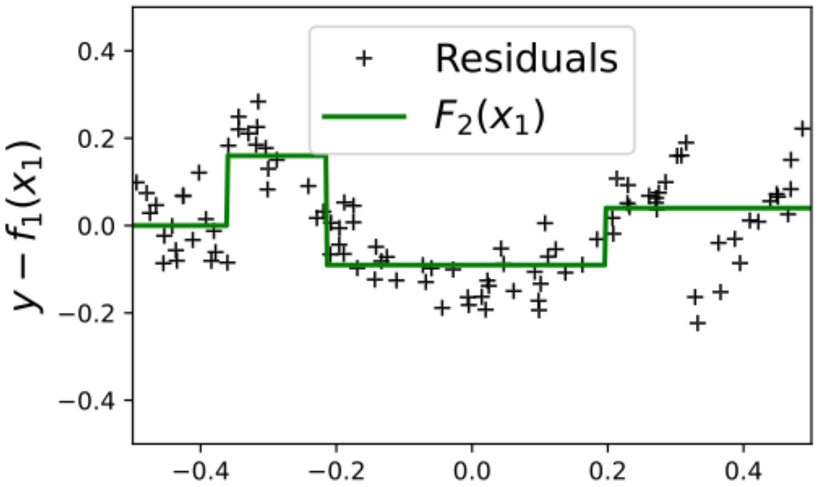

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)

y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)

y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

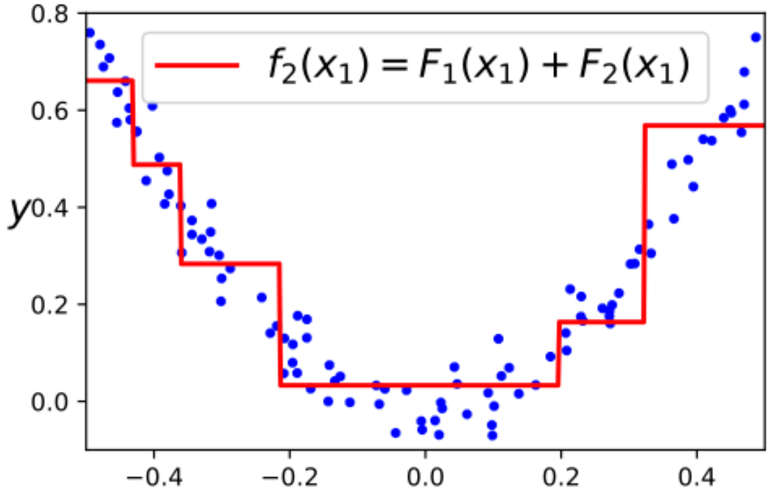
Quadratic Loss and Least Squares Boosting

Training Set:

$y = 3x^2 + 0.05z$, where $z \sim N(0,1)$

$(y_i, x_i), i = 1, \ldots, N$





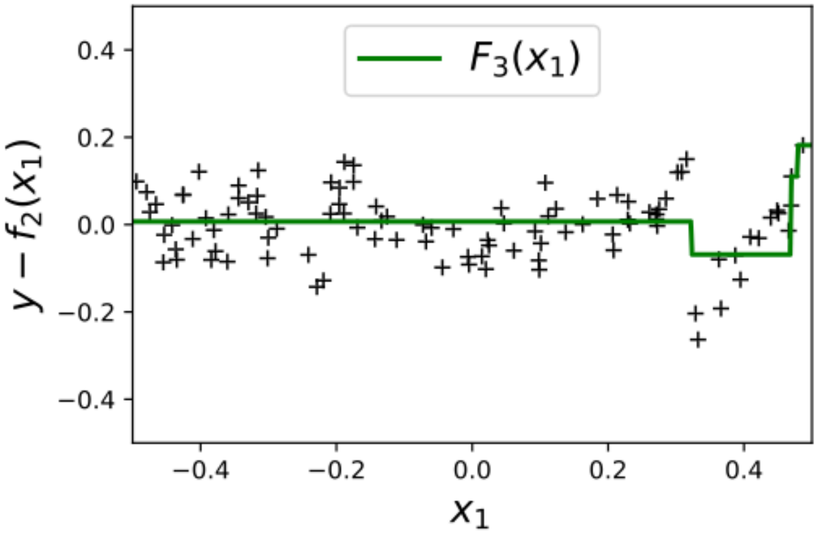

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)

y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)

y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

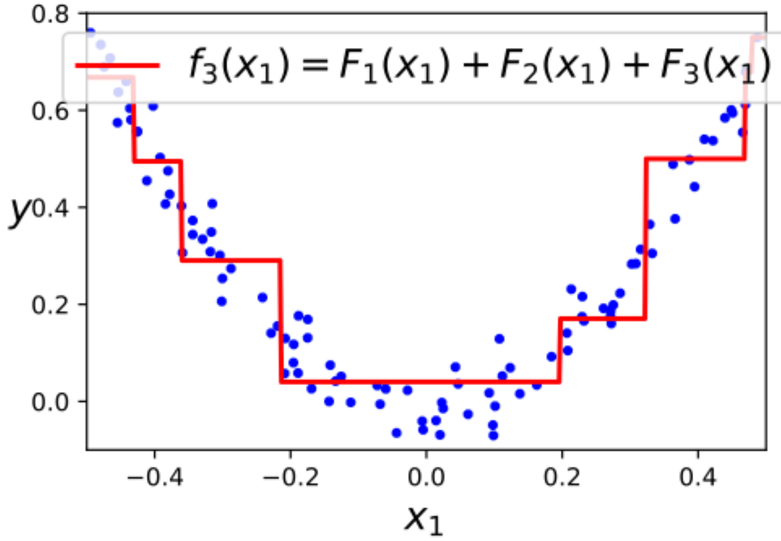
Quadratic Loss and Least Squares Boosting

Training Set:

$y = 3x^2 + 0.05z$, where $z \sim N(0,1)$

$(y_i, x_i), i = 1, \ldots, N$






```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)

y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)

y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

AdaBoost (Additive Boosting)

Classification setting: $(x_i, y_i) \in (X, \pm 1)$

Normally each data-point has a weight $1/N$

AdaBoost modifies the weights to $w_i > 0$ with $\sum_{i=1}^{N} w_i = 1$

First iteration: $w_i = 1/N$

Correct classified points have their weights decreased

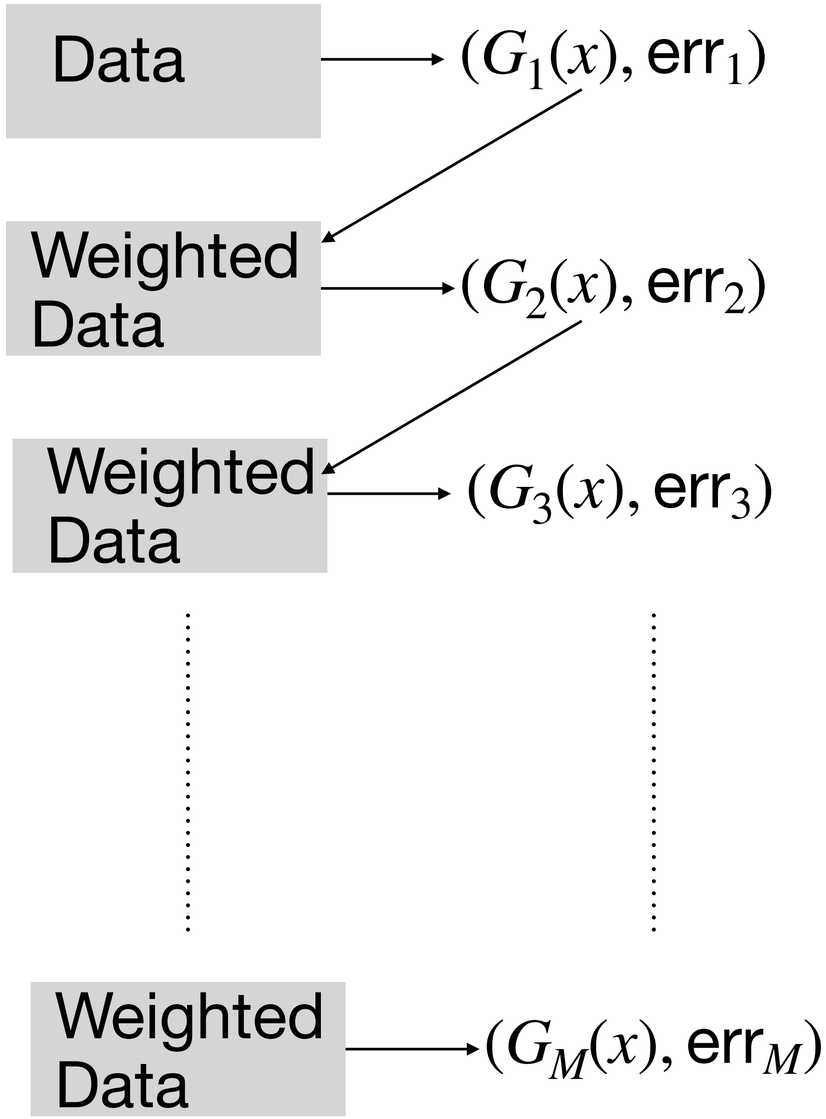
Incorrect points have their weights increased

Final output is a weighted sum of all classifiers (the good ones get a higher weight)

AdaBoost (Additive Boosting)

Classification setting: $(x_i, y_i) \in (X, \pm 1)$

Error of classifier, $H(x)$ is err $= \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i = H(x_i))$



Final output is a weighted combination of all classifiers

Classifiers with low error get higher weight

AdaBoost M.1

1. Initial weights: $w_i = 1/N$

2. For $m = 1,\ldots,M$

    2.1. Compute error:

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}(y_i \neq G(x_i))}{\sum_{i=1}^{N} w_i} \in [0,1]$$

    2.3. Compute weight of classifier:

$$\alpha_m = \log\left((1 - \text{err}_m)/\text{err}_m\right)$$

    2.4. Reweight data-points: $w_i \leftarrow w_i \exp\left(\alpha_m \mathbb{I}(y_i \neq G_m(x_i)\right),\ i = 1,\ldots,N$

3. Final output: $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x_i)\right)$

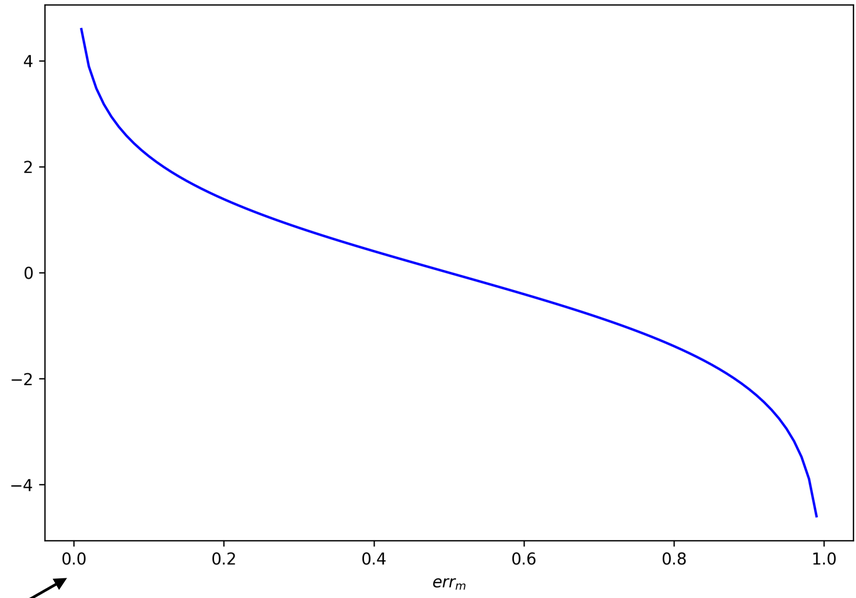
AdaBoost M.1

1. Initial weights: $w_i = 1/N$

2. For $m = 1,\ldots,M$

   2.1. Compute error:

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}(y_i \neq G(x_i))}{\sum_{i=1}^{N} w_i} \in [0,1]$$



   2.3. Compute weight of classifier:

$$\alpha_m = \log\left((1 - \text{err}_m)/\text{err}_m\right)$$

   2.4. Reweight data-points: $w_i \leftarrow w_i \exp\left(\alpha_m \mathbb{I}(y_i \neq G_m(x_i))\right),\ i = 1,\ldots,N$

3. Final output: $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x_i)\right)$

AdaBoost and Stage-Wise Additive Modelling

Stage-wise additive modelling for:

$$\min_{\gamma,\beta_m} \sum_{i=1}^{N} L\left(y_i, \sum_{i=1}^{M} \beta_m b_m(x_i, \gamma_m)\right)$$

1. Start with $f_0(x)$
2. For $m = 1,\ldots M$:

2.1 Solve: $(\beta_m, \theta_m) \in \arg\min_{\theta,\beta} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + \beta F_m(x_i, \theta)\right)$

2.2 Update: $f_m(x) = f_{m-1}(x) + \beta_m F(x, \gamma_m)$

How is this related to AdaBoost?

Exponential Loss and AdaBoost

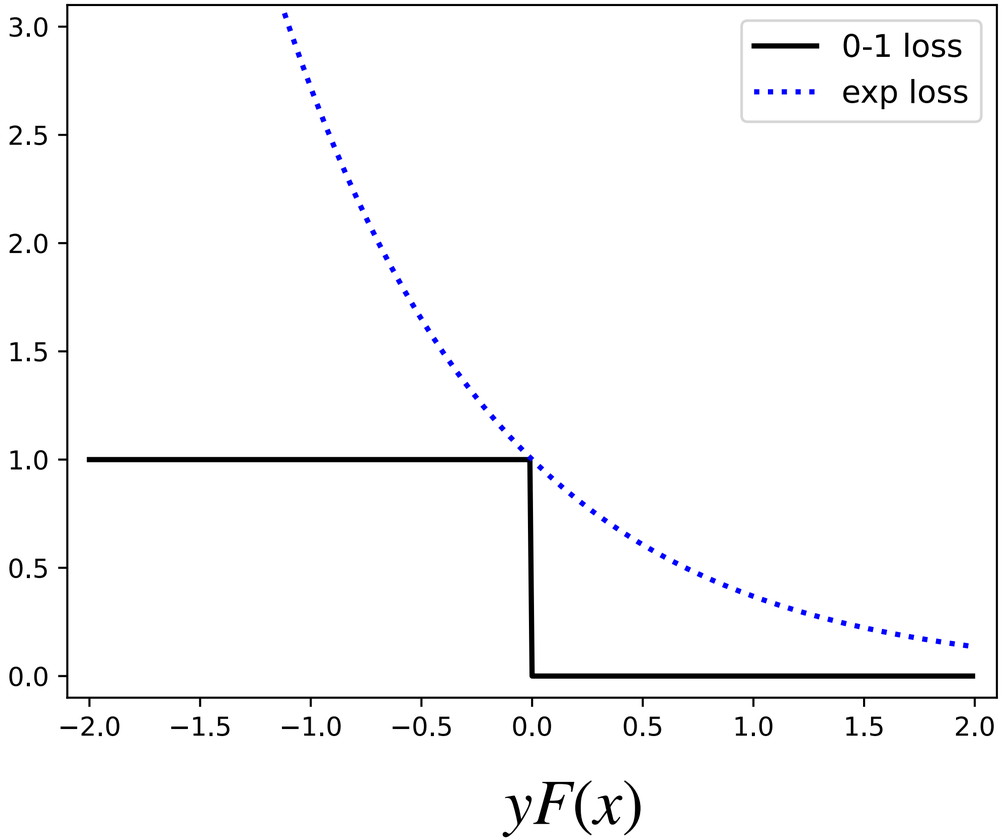
Classification setting: $(x_i, y_i) \in (X, \pm 1)$

Exponential Loss function: $L(y, F(x)) = \exp(-yF(x))$

0/1 Loss function: $L(y, F(x)) = \begin{cases} 1 \text{ if } yF(x) < 0 \\ 0 \text{ otherwise} \end{cases}$

The exponential loss is a smooth upper bound for 0/1 so AdaBoost uses it as the loss function

The various steps of the algorithm are merely a way to solve the optimisation problem!



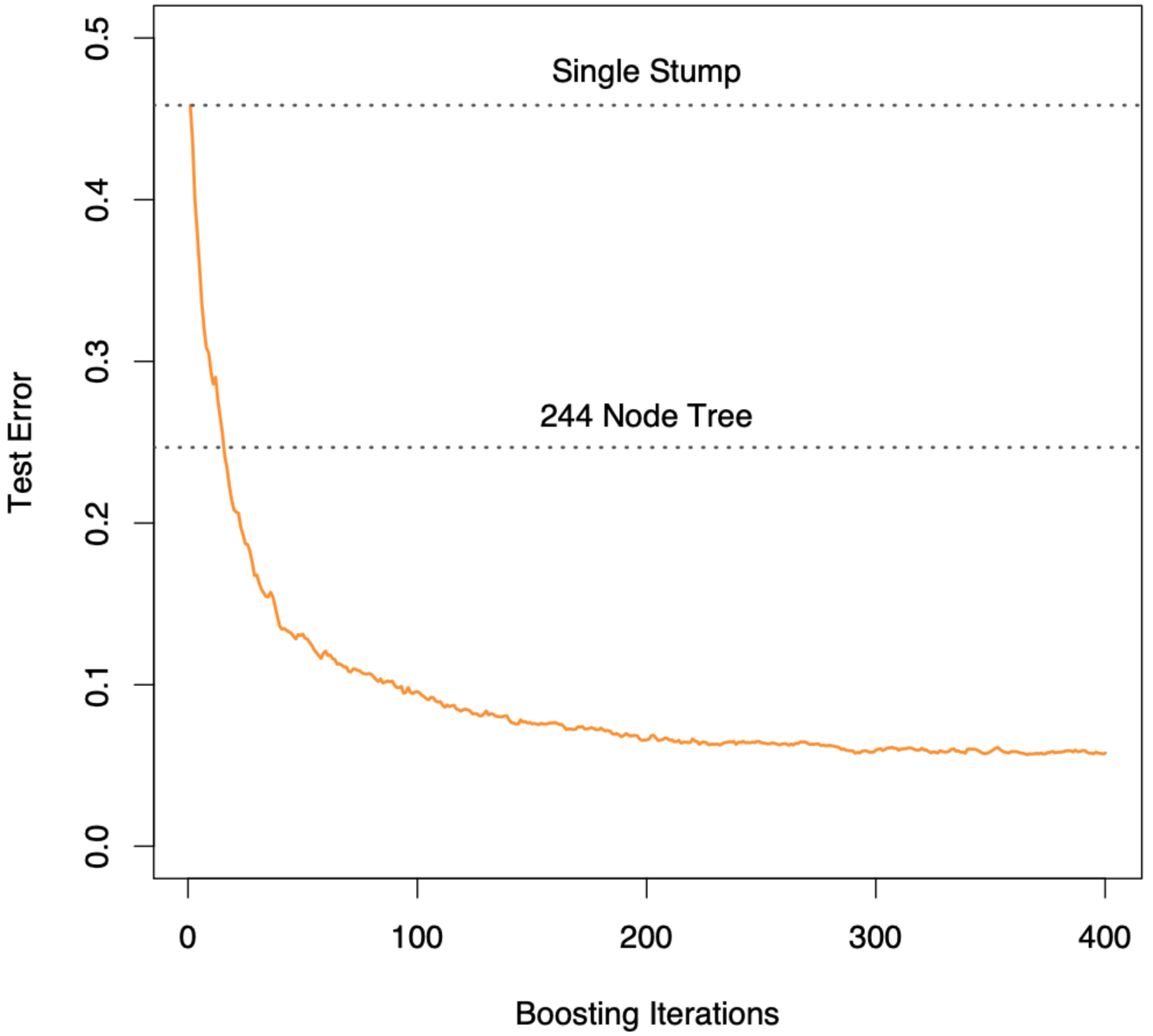
Demo

$$Y = \begin{cases} 1 & \text{if } \sum_{j=1}^{10} X_j < 9.5 \\ -1 & \text{otherwise} \end{cases}$$

where $X_j \sim N(0,1)$

stump is a decision tree with depth=1

Note: the 244 node tree has a bias problem resolved via boosting



Variations

Boosting is a general strategy, and AdaBoost has many variants.

- **Real AdaBoost**: Decision tress output class probability. Real AdaBoost uses this probability in the optimisation problem (usually works better)
- **LogitBoost**: Minimises a logistic loss function
- **Brownboost**: Applicable for problems with noise, it “gives up” on data points that are repeatedly misclassified.
- **Gentleboost**: Instead of choosing the step size that improves the objective as much as possible, Gentleboost uses a fixed step-size (which tends to generalise better).

There are others!

Motivation for Gradient Boosting

Instead of deriving a new algorithm for each loss function, Gradient Boosting just solves the optimisation problem iteratively.

Consider the optimisation over all possible learning functions f:

$$f^{\star} = \arg\min_{f} L(f)$$

Since f cannot be easily represented, we just represent it at:

$f = [f(x_1), \ldots, f(x_N)]$ at the training points

We then do **gradient descent**: $f_m = = f_{m-1} - \beta_m g_m$

Gradient Boosting: Gradients and Stepsizes

The key step in Gradient Boosting: $f_m = = f_{m-1} - \beta_m g_m$

where $g_m = \dfrac{\partial L(y_i, f(x_i))}{\partial f(x_i)}$ evaluated at $f = f_{m-1}$

Examples:

Name	Loss	$\partial L(y_i, f(x_i))/\partial f(x_i)$
Squared Error	$0.5(y_i - f(x_i))^2$	$y_i - f(x_i)$
Absolute Error	$\lvert y_i - f(x_i)\rvert$	$\text{sign}(y_i - f(x_i))$
Exponential Loss	$\exp(-y_i f(x_i))$	$-y_i \exp(-y_i f(x_i))$

β_m can be selected via line-search $\beta_m = \arg\min_{\beta} L(f_{m-1} - \beta g_m)$

Gradient Boosting Avoiding Overfitting

If we simply implemented: $f_m = = f_{m-1} - \beta_m g_m$

The algorithm will overfit, instead, Gradient Boosting fits a weak learner to the negative gradient

$$F_m = \arg\min_F \sum_{i=1}^{N} (-g_{im} - F(x_i))^2$$

e.g. Fit a CART tree with data, (x_i, g_{im}) to compute F_m

if the loss is the square loss we recover L2Boosting

Extreme Gradient Boosting XGBoost

XGBoost is a very efficient and widely used implementation of gradient boosted trees

Main features:

- Adds a regulariser to help with overfitting
- Uses second derivatives for the loss (instead of just the gradient)
- It samples features (like random forests)
- Very efficient implementations are available (CatBoost, LightGBM)

Gradient Boosted Trees — Other names

Depending on the implementation details, **gradient boosting** has different names:

- **Gradient Boosting Machine** (GBM)
- **Functional Gradient Boosting**
- **Multiple Additive Regression Trees** (MART)
- **Boosted Regression Trees** (BRT).
- **XGBoost**

Interpreting Tree Ensembles

A big advantage of decision trees is that they are interpretable

However after boosting, bagging etc they are not as easy to interpret

A key measure to understand what the tree learned is **feature importance**:

$$R_k(T) = \sum_{j=1}^{J-1} G_j \mathbb{I}(v_j = k)$$

Single Tree

Feature Importance of feature k for Tree T

$G_j =$ reduction in cost at node j

$v_j = k$ if node j uses feature k

$$R_k(T_1, T_2, \ldots T_M) = \frac{1}{M} \sum_{m=1}^{M} R_k(T_m)$$

M Trees

Empirical Asset Pricing via Machine Learning*

$$r_{i,t+1} = g(z_t) + \eta_t$$

Noise

Predict Asset return

Factors/Predictor Variables (e.g. momentum, liquidity etc..)

Evaluated several ML methods.

*S Gu, B. Kelly, D. Xiu, **Empirical Asset Pricing via Machine Learning**, The Review of Financial Studies, Volume 33, Issue 5, May 2020, Pages 2223–2273

Empirical Asset Pricing via Machine Learning*

$$r_{i,t+1} = g(z_t) + \eta_t$$

Noise

Predict Asset return

Factors/Predictor Variables (e.g. momentum, liquidity etc..)

Evaluated several ML methods.

Conclusion: Decision Trees are very competitive

*S Gu, B. Kelly, D. Xiu, **Empirical Asset Pricing via Machine Learning**, The Review of Financial Studies, Volume 33, Issue 5, May 2020, Pages 2223–2273

Machine Learning for Stock Selection*

- Stock selection as a classification problem
- Label: Outperforming vs underperforming stocks
- Overfitting is a big problem in financial data sets
- Combination of various weak learners (SVMs, NNs) can be helpful

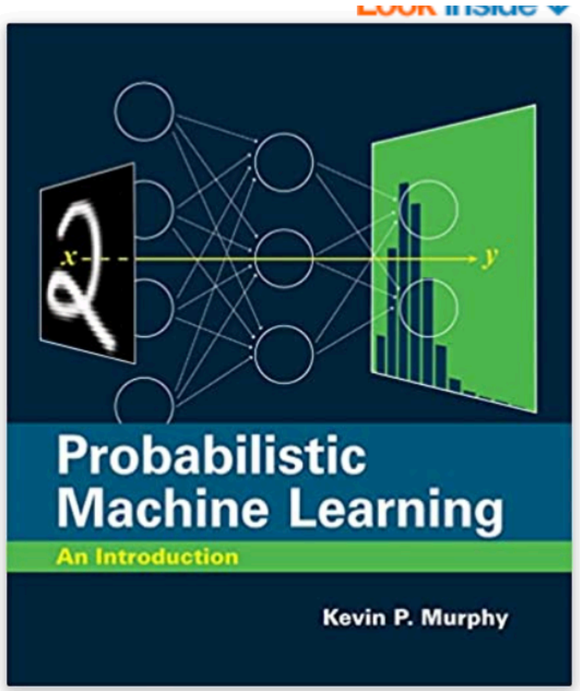
*K.C. Rasekhschaffe & Robert C. Jones ,**Machine Learning for Stock Selection**, Financial Analysts Journal 2019

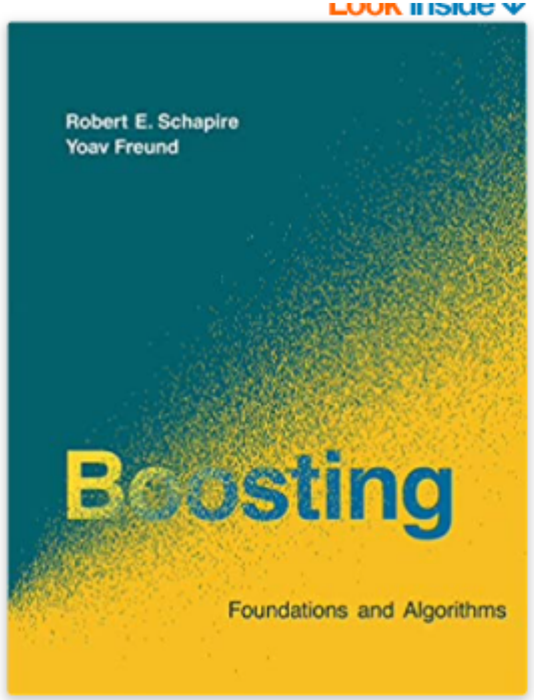
Statistical Arbitrage on the S&P500*

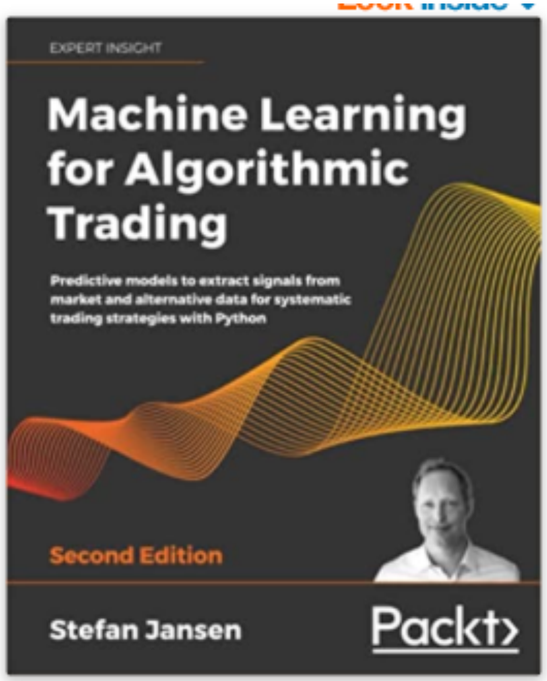
- Study of deep neural networks, gradient-boosted-trees, random forests (RAF) for statistical arbitrage.
- Generate daily one-day-ahead trading signals on the probability to outperform the market.
- Promising results for ensemble method (average of DNN, a gradient boosted tree and a random forest)

C. Krauss, X. A. Do, N. Huck, **Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500**, European Journal of Operational Research, 2017

Some References







Conclusions

- Decision Trees are an **extremely powerful** ML algorithm
- They are **easy to apply and explain** (no maths required!)
- A standard CART tree is unlikely to work
- Figure out if your tree has a variance or a bias problem
- Address the problem with **ensemble training or boosting**