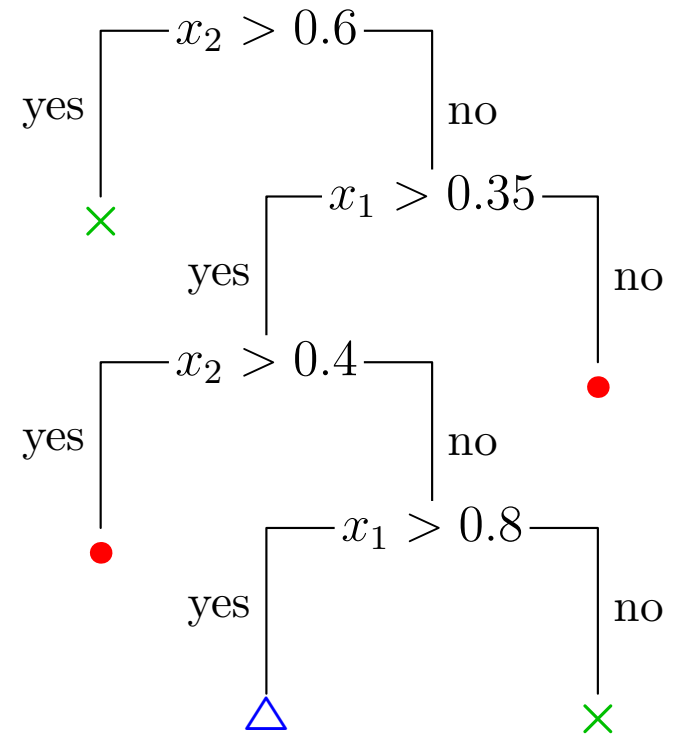
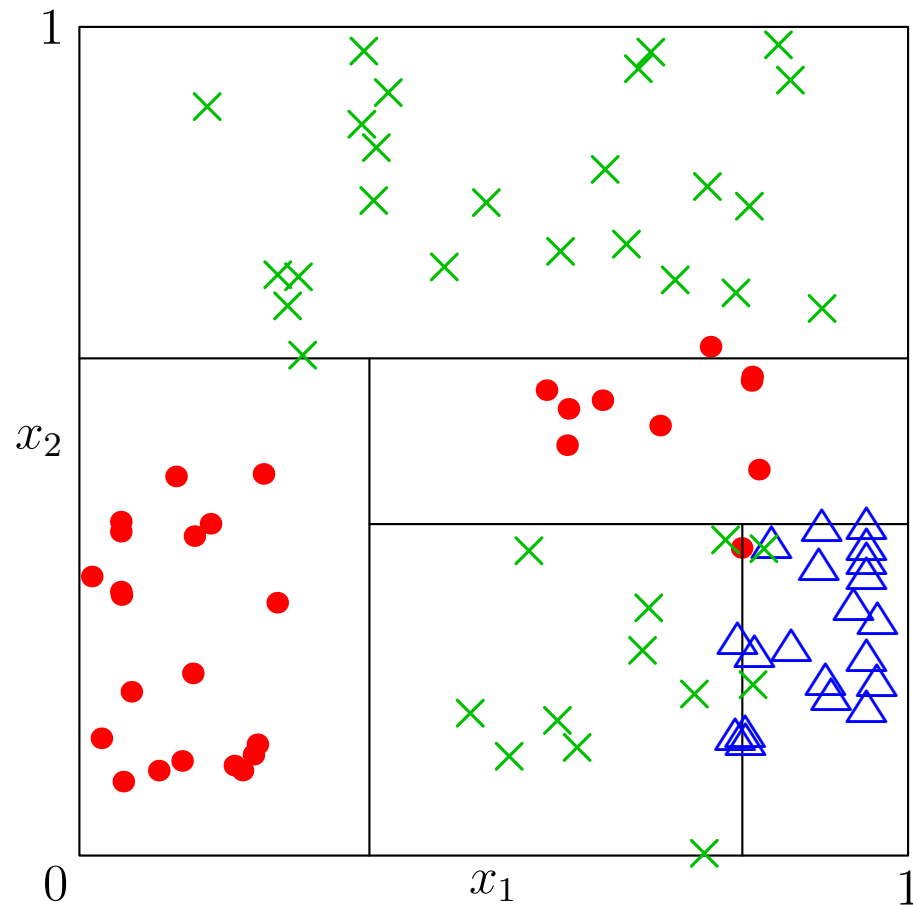


# Advanced Machine Learning

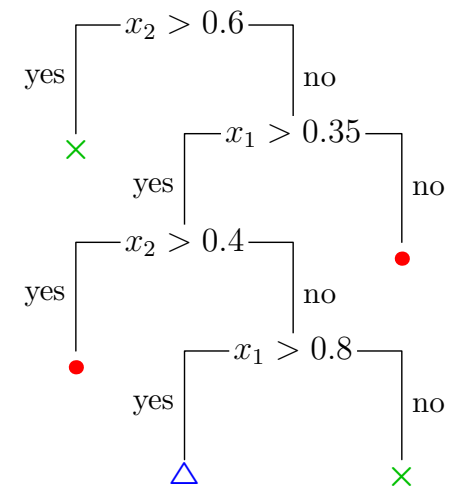
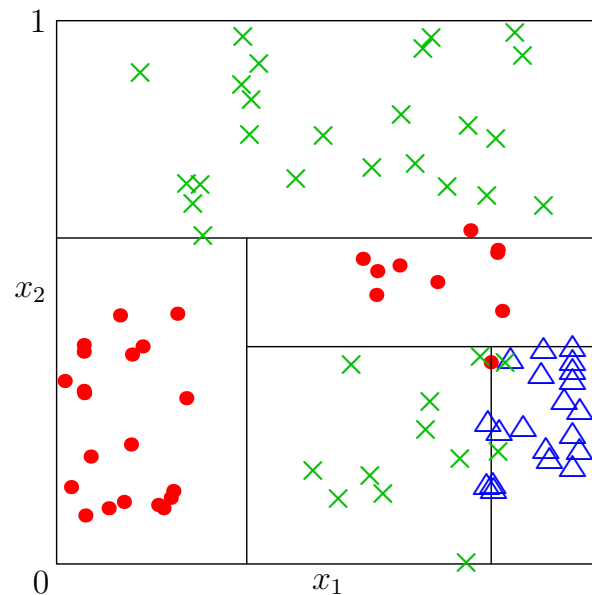
## *Boosting*



*Boosting, AdaBoost, Gradient Boosting*

# Outline

1. **Boosting**
2. AdaBoost
3. Gradient Boosting



# Boosting

- In boosting we make a **strong learner** by using a weighted sum of **weak learners**

$$C_n(\mathbf{x}) = \sum_{i=1}^n \alpha_i \hat{h}_i(\mathbf{x})$$

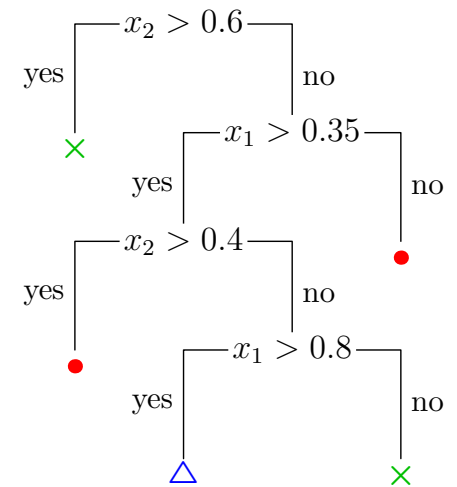
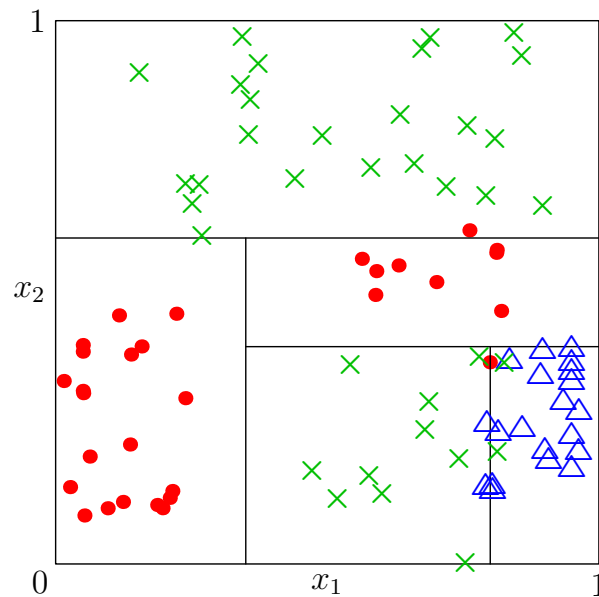
- Weak learners,  $\hat{h}_i(\mathbf{x})$ , are learning machine that do a little better than chance
- The trick is to choose the weights,  $\alpha_i$
- Because the weak learners do little better than chance we (miraculously) **don't** overfit that much

# Shallow Trees

- One of the most effective type of weak learner are very shallow trees■
- Sometimes we just use one variable (the stump)■, although usually we would use slightly deeper trees■
- There are different algorithms for choosing the weights
  - ★ adaboost■—a classic algorithm for binary classification■
  - ★ gradient boosting■—used for regression, trains a weak learner on the residual errors■

# Outline

1. Boosting
2. **AdaBoost**
3. Gradient Boosting



# Boosting a Binary Classifier

- Suppose we have a binary classification task with data  $\mathcal{D} = \{(\mathbf{x}^\mu, y^\mu) | \mu = 1, 2, \dots, m\}$  with  $y^\mu \in \{-1, 1\}$ ■
- Our  $i^{th}$  weak learner provides a prediction  $\hat{h}_i(\mathbf{x}^\mu) \in \{-1, 1\}$ ■
- We ask, can we find a linear combination

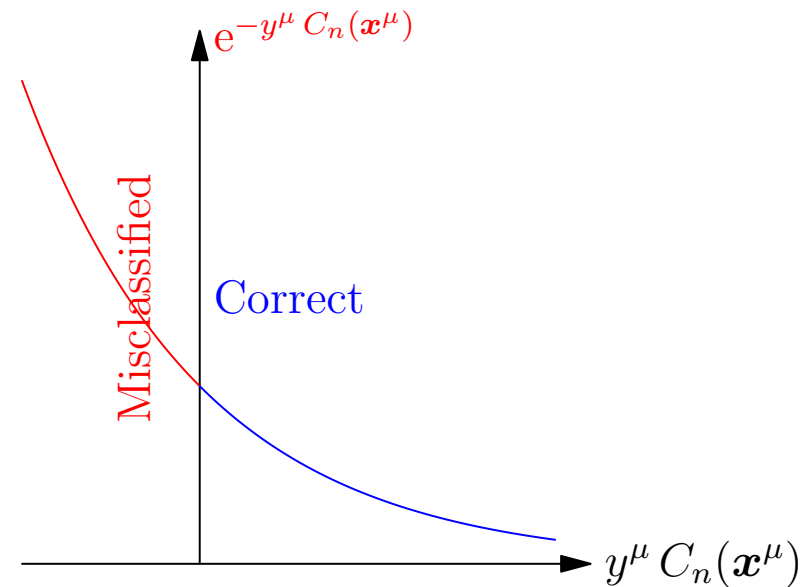
$$C_n(\mathbf{x}) = \alpha_1 \hat{h}_1(\mathbf{x}) + \alpha_2 \hat{h}_2(\mathbf{x}) + \dots + \alpha_n \hat{h}_n(\mathbf{x})$$

- So that  $\text{sgn}(C_n(\mathbf{x}))$  is a strong learner?■
- Note we want  $y^\mu C_n(\mathbf{x}^\mu) > 0$ ■

# AdaBoost

- AdaBoost is a classic solution to this problem■
- It assigns an “loss function”

$$L_n = \sum_{\mu=1}^m e^{-y^\mu C_n(\mathbf{x}^\mu)} \blacksquare$$



- This punishes examples where there is an errors more than correct classifications■

# Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\mathbf{x}) = C_{n-1}(\mathbf{x}) + \alpha_n \hat{h}_n(\mathbf{x})$$

- Defining  $w_1^\mu = 1$  and  $w_n^\mu = e^{-y^\mu C_{n-1}(\mathbf{x}^\mu)}$  then

$$\begin{aligned} L_n(\alpha_n) &= \sum_{\mu=1}^m e^{-y^\mu C_n(\mathbf{x}^\mu)} = \sum_{\mu=1}^m e^{-y^\mu (C_{n-1}(\mathbf{x}^\mu) + \alpha_n \hat{h}_n(\mathbf{x}^\mu))} \\ &= \sum_{\mu=1}^m w_n^\mu e^{-\alpha_n y^\mu \hat{h}_n(\mathbf{x}^\mu)} = e^{\alpha_n} \sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu e^{-\alpha_n} + e^{-\alpha_n} \sum_{\mu: y^\mu = \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu \\ &= e^{-\alpha_n} \sum_{\mu=1}^m w_n^\mu + (e^{\alpha_n} - e^{-\alpha_n}) \sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu \end{aligned}$$



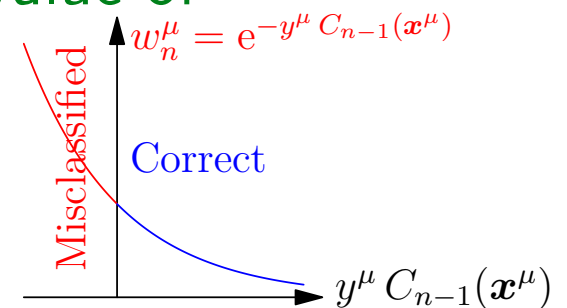
# Choosing a Weak Classifier

- To minimise the loss

$$L_n(\alpha_n) = e^{-\alpha_n} \sum_{\mu=1}^m w_n^\mu + (e^{\alpha_n} - e^{-\alpha_n}) \sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu$$

- We choose the weak learner with the lowest value of

$$\sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu = \sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} e^{-y^\mu C_{n-1}(\mathbf{x}^\mu)}$$



- That is, it misclassifies only where the other learners classify well

# Choosing Weights

- We now choose the weight  $\alpha_n$  to minimise the loss  $L_n(\alpha_n)$

$$\frac{\partial L_n(\alpha_n)}{\partial \alpha_n} = e^{\alpha_n} \sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu - e^{-\alpha_n} \sum_{\mu: y^\mu = \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu = 0 \quad \blacksquare$$

- That is

$$e^{2\alpha_n} = \frac{\sum_{\mu: y^\mu = \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu}{\sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu} \quad \blacksquare \quad \text{or} \quad \alpha_n = \frac{1}{2} \log \left( \frac{\sum_{\mu: y^\mu = \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu}{\sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu} \right) \quad \blacksquare$$

# Algorithm

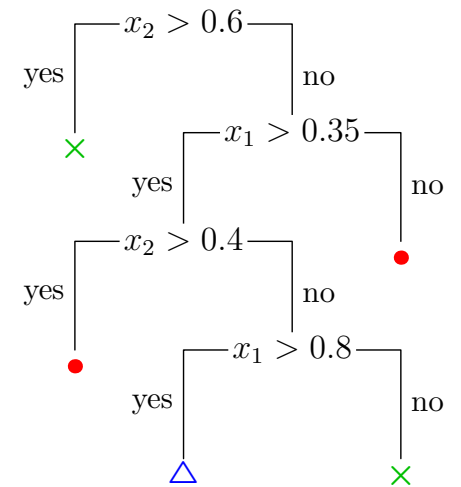
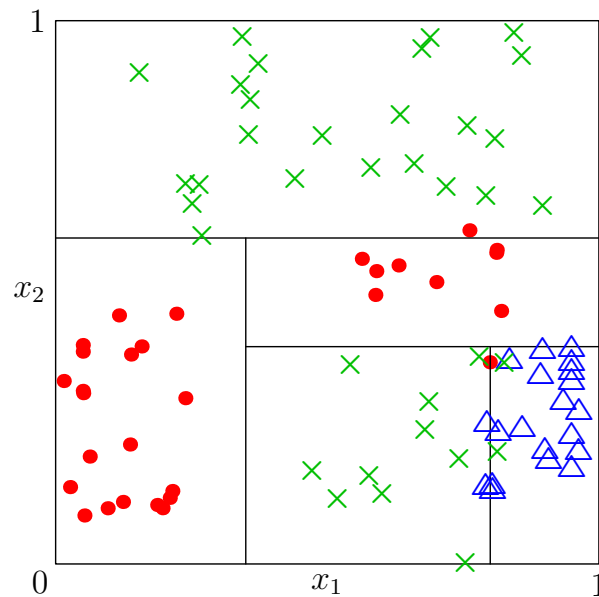
1. Start with a set of weak learners  $\mathcal{W}$ ■
2. Associate a weight,  $w_n^\mu$ , with every data point  $(\mathbf{x}^\mu, y^\mu)$ ,  $\mu = 1, 2, \dots, m$ ■
3. Initially  $w_1^\mu = 1$ ■ (large weight,  $w_n^\mu$ , means  $(\mathbf{x}^\mu, y^\mu)$  is poorly classified)■
4. Choose the weak learning,  $\hat{h}_n(\mathbf{x}) \in \mathcal{W}$ , that minimises 
$$\sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu$$
■
5. Update predictor  $C_n(\mathbf{x}) = C_{n-1}(\mathbf{x}) + \alpha_n \hat{h}_n(\mathbf{x})$  where 
$$\alpha_n = \frac{1}{2} \log \left( \frac{\sum_{\mu: y^\mu = \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu}{\sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu} \right)$$
■
6. Update  $w_{n+1}^\mu = w_n^\mu e^{-y^\mu \alpha_n \hat{h}_n(\mathbf{x}^\mu)}$ ■
7. Go to 4■

# Performance

- Adaboost works well with weak learners, usually out-performing bagging■
- It doesn't work well with strong learners (tends to over-fit)■
- It is limited to binary classification (there are generalisation, but they are difficult to get to work)■
- It has fallen from fashion■
- In contrast **gradient boosting** used for regression is very popular■

# Outline

1. Boosting
2. AdaBoost
3. **Gradient Boosting**



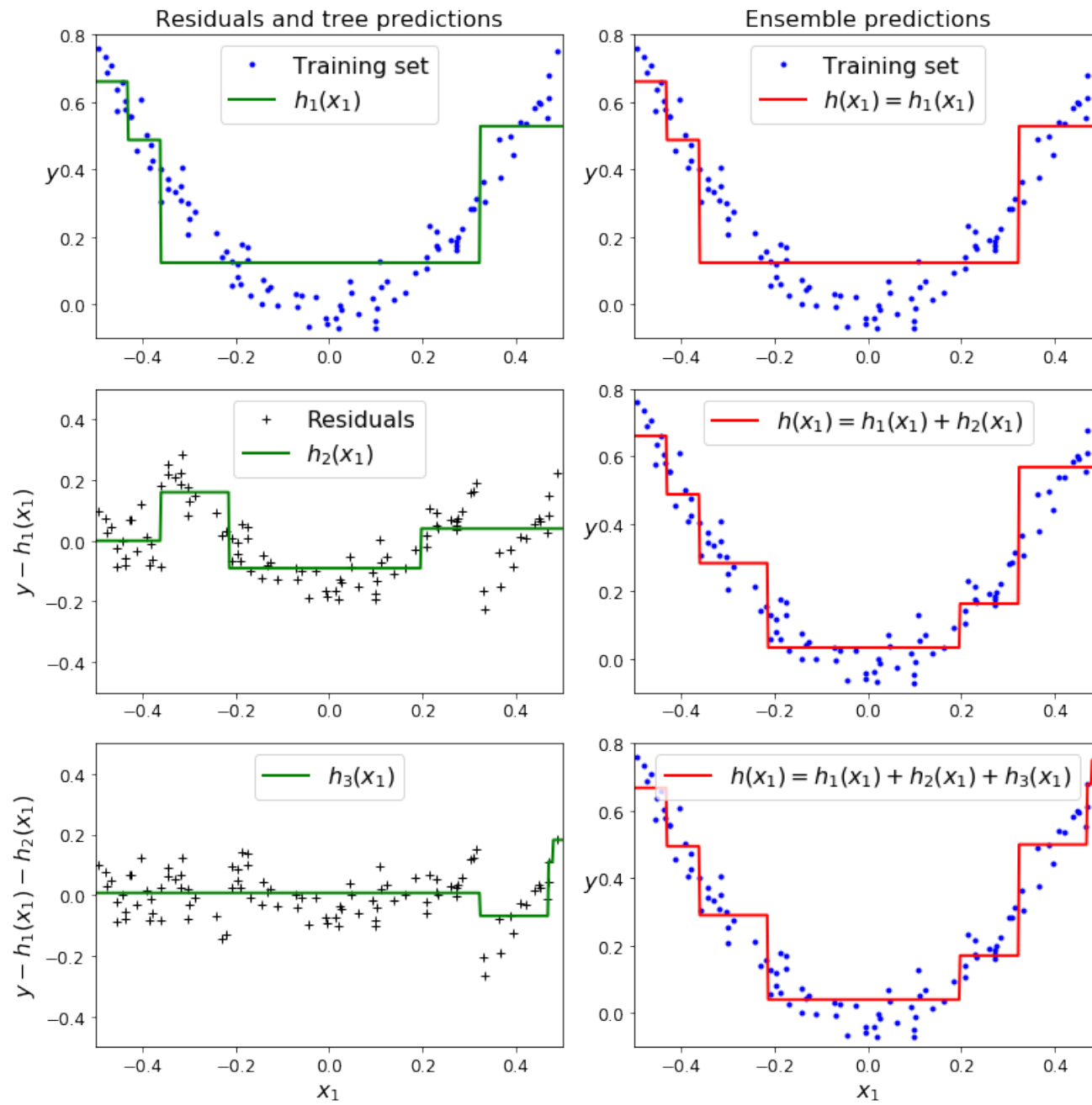
# Gradient Boosting

- In gradient boosting we again build a strong learner as a linear combination of weak learners

$$C_n(\mathbf{x}) = C_{n-1}(\mathbf{x}) + \hat{h}_n(\mathbf{x}) \blacksquare$$

- Gradient boosting used on regression (again using decision trees)  $\blacksquare$
- At each step  $\hat{h}_n(\mathbf{x})$  is trained to predict the residual error,  $\Delta_{n-1} = y - C_{n-1}(\mathbf{x})$ , (i.e. the target minus the current prediction)  $\blacksquare$
- (This difference looks a bit like a gradient hence the rather confusing name)  $\blacksquare$

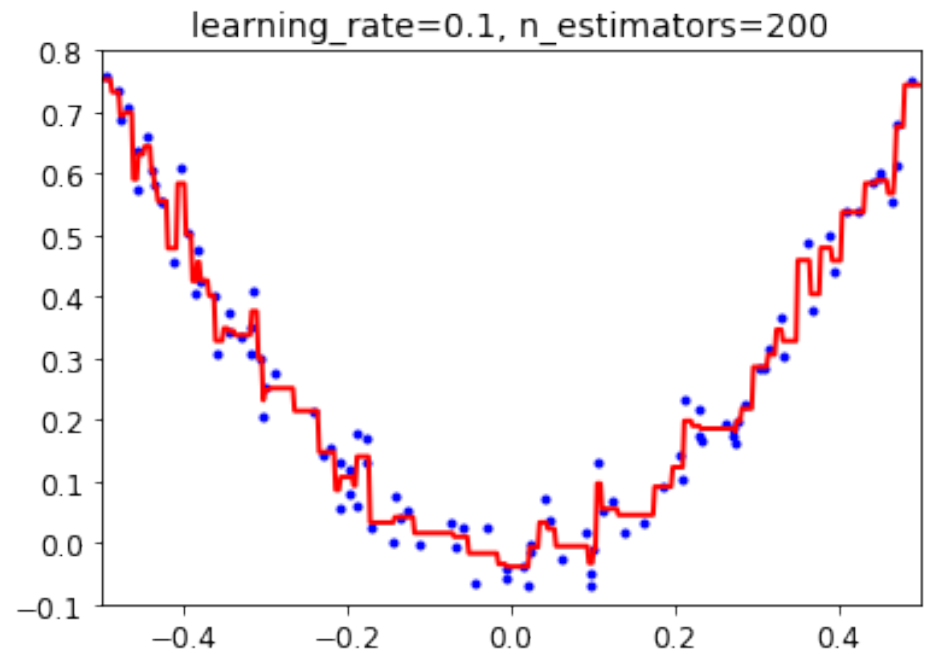
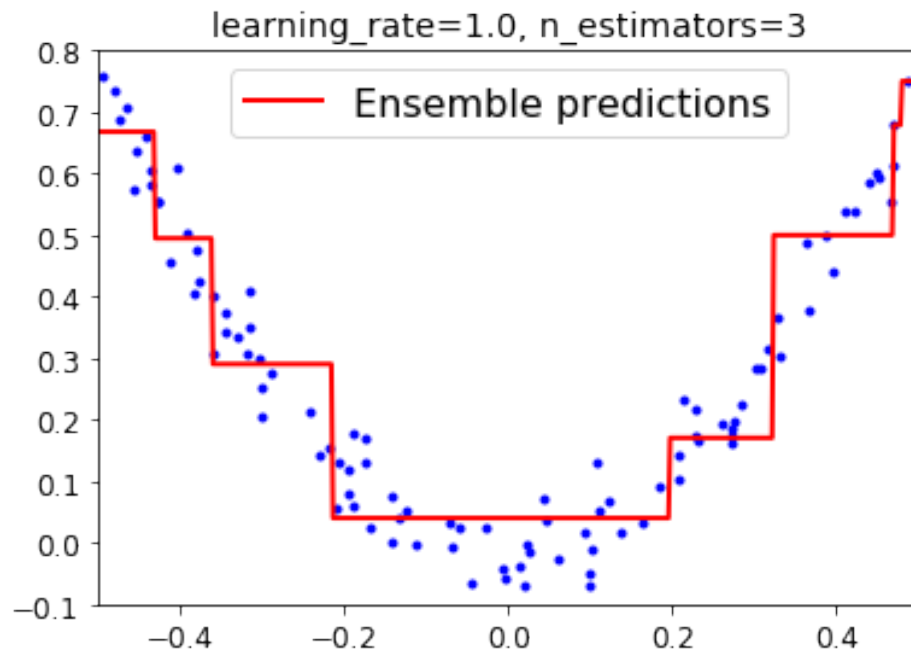






# Keep On Going

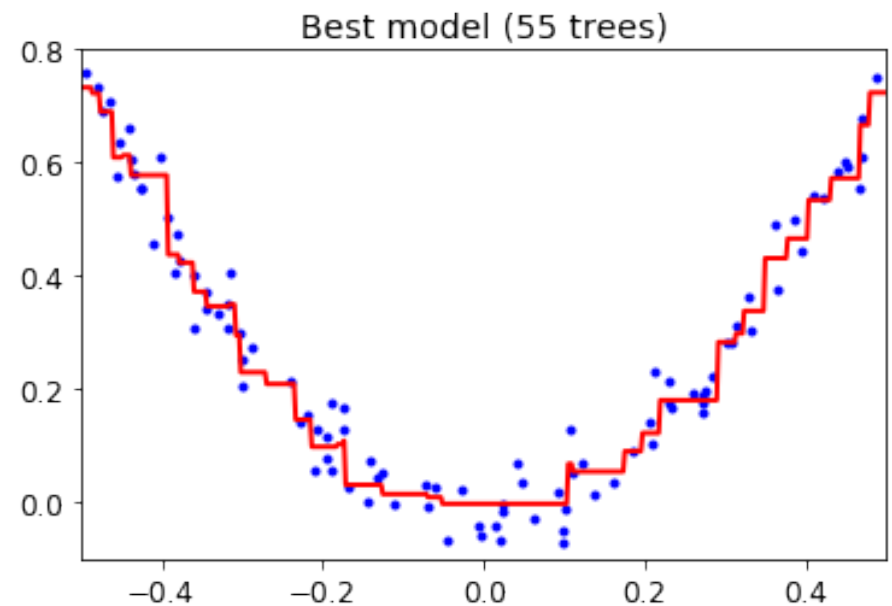
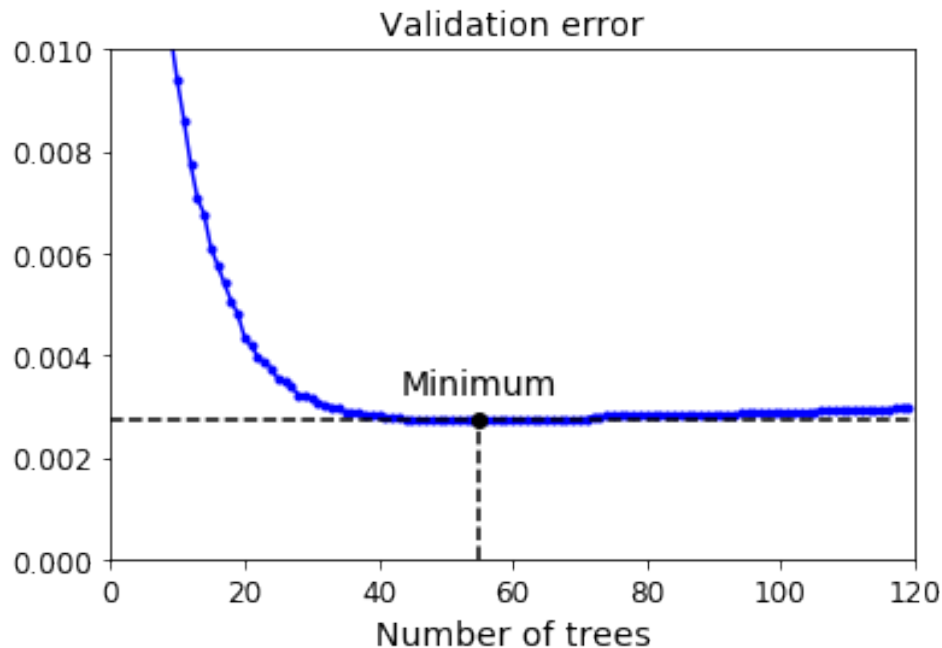
- We can keep on going



- But we will over-fit eventually

# Early Stopping

- Like many algorithms we often get better results by early stopping



- Use cross-validation against a validation set to decide when to stop

# XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions■
- XGBoost stands for eXtreme Gradient Boosting■
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better■
- It uses a cleverly chosen regularisation term to favour simple trees■
- Finds a clever way to approximately minimise error plus regulariser very fast■
- Rather a bodge of optimisation hacks■

# Conclusion

- Ensemble methods have proved themselves to be very powerful■
- Tend to work best with very simple models (true of random forest and boosting)■—seems to reduce over-fitting■
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)■—probably■
- For images, signal and speech deep learning can give very significant advantage■
- Probabilistic models can be better if you have a good model■