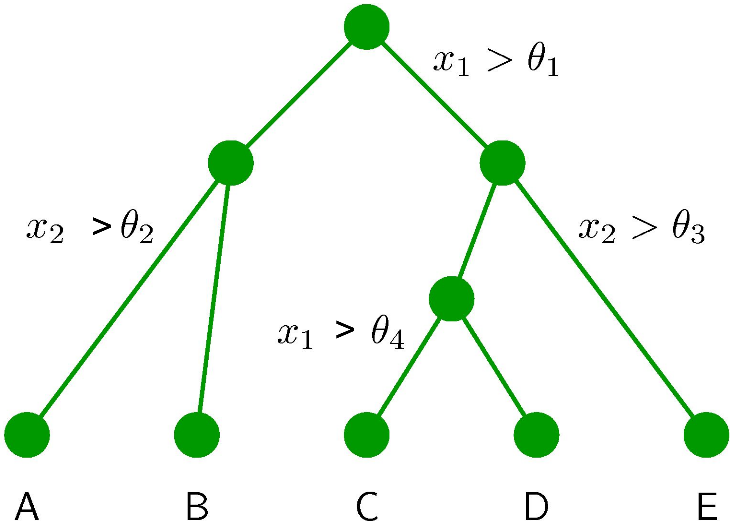
Ensemble Methods

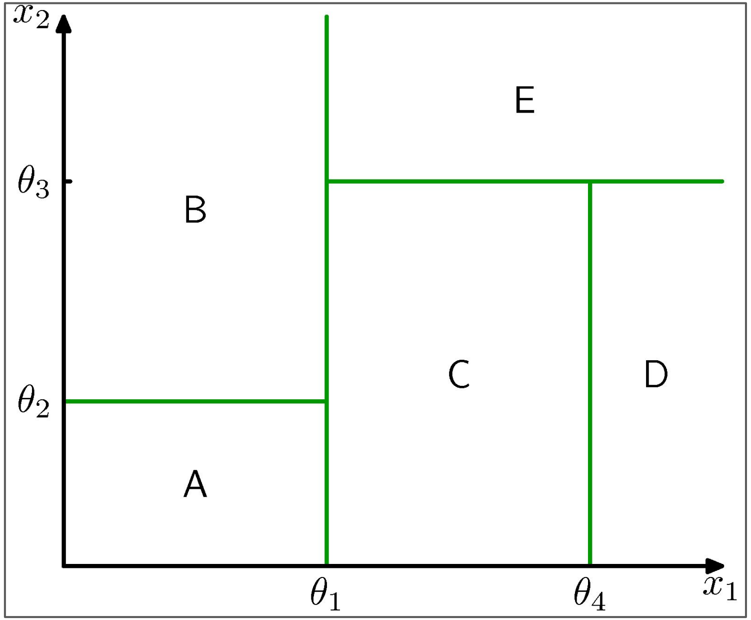
CS 580
Intro to Artificial Intelligence

Some notes about Decision Trees

- Decision Trees can work for regression
  - Output is the average of the values in the leaf (other choices work too)
- DT’s can work with continuous features
  - Need a new method to pick “questions” (datapoint values, midpoints, random, etc…)
  - Splits the input space into (axis aligned?) boxes
- The complexity of DT’s grows with depth, *extremely* quickly (2^2^D)
  - Might be difficult to pick a good maximum depth: depth d might underfit, but d+1 might overfit
- Two techniques for controlling complexity with a little more nuance
  - Bagging: build a bunch of overfit trees, then “smooth” the result
  - Boosting: start with a single underfit tree, then iteratively improve by adding more trees
- ML techniques that use a collection of learners: **Ensemble** methods
- DT variants that use a collection of trees: **Forests**

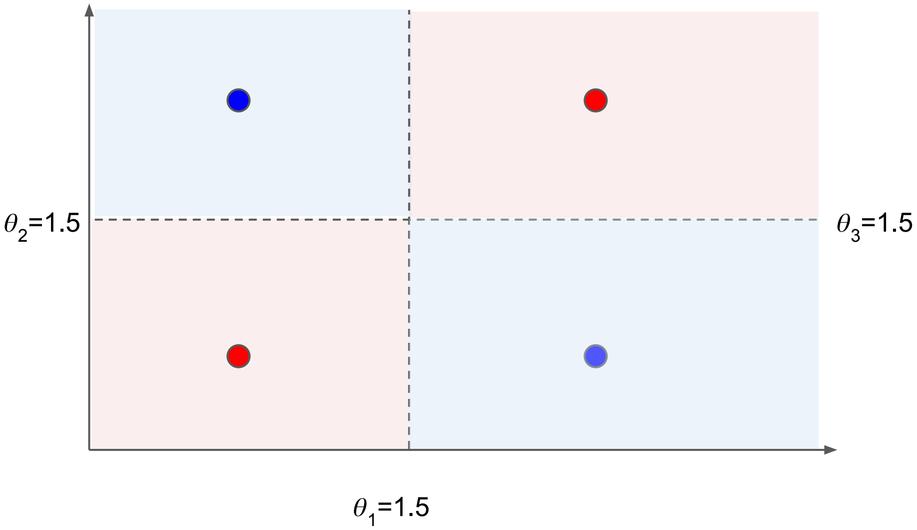
DT's for continuous input spaces

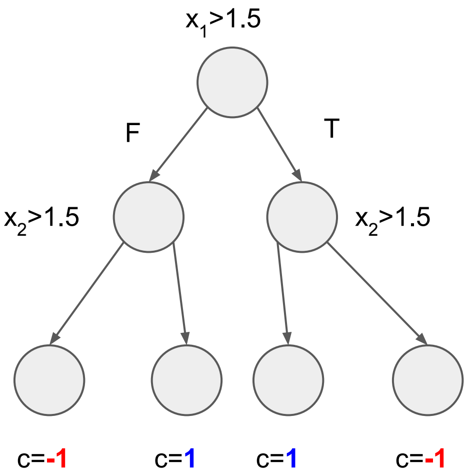




Simple example

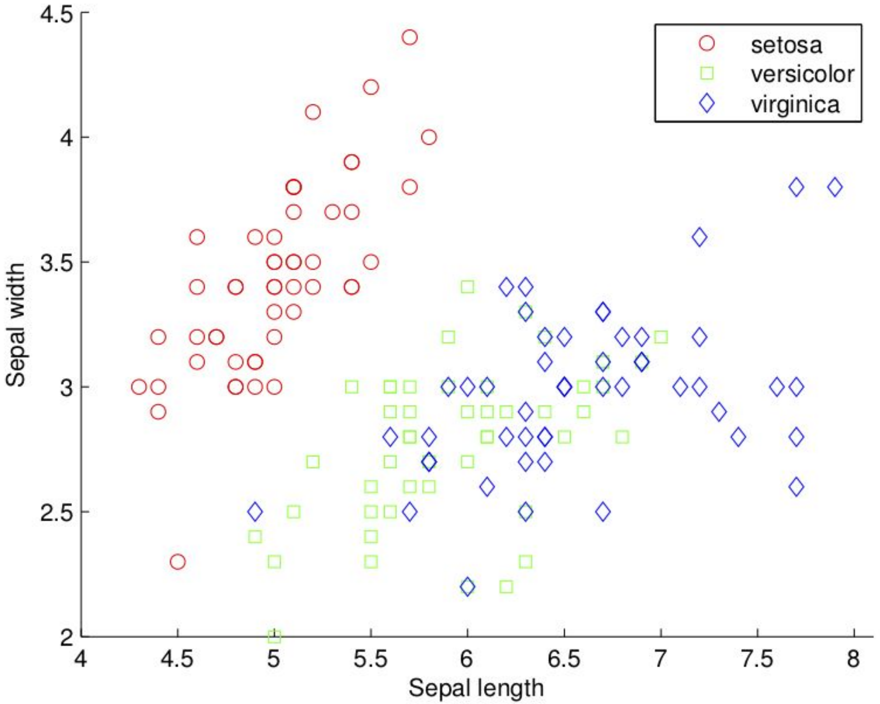
x	y
(1,1)	-1
(1,2)	1
(2,1)	1
(2,2)	-1

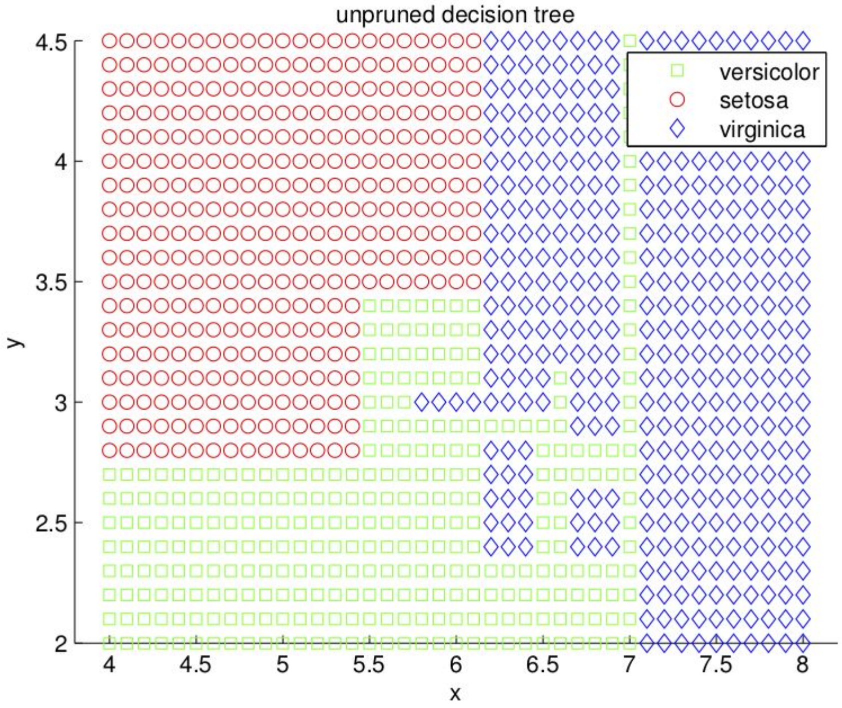




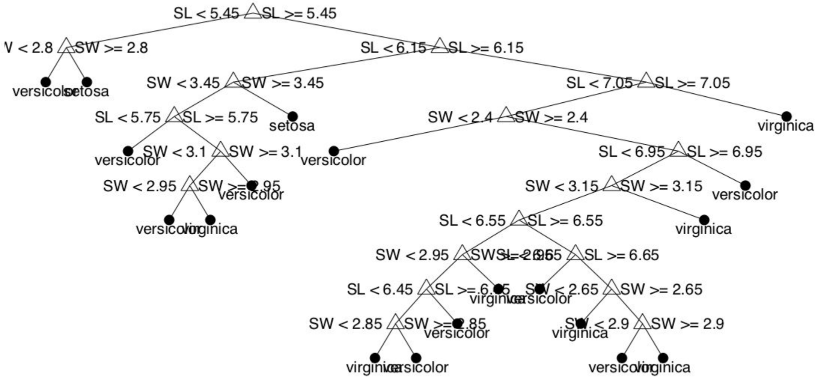
Question: did we need 3 layers?

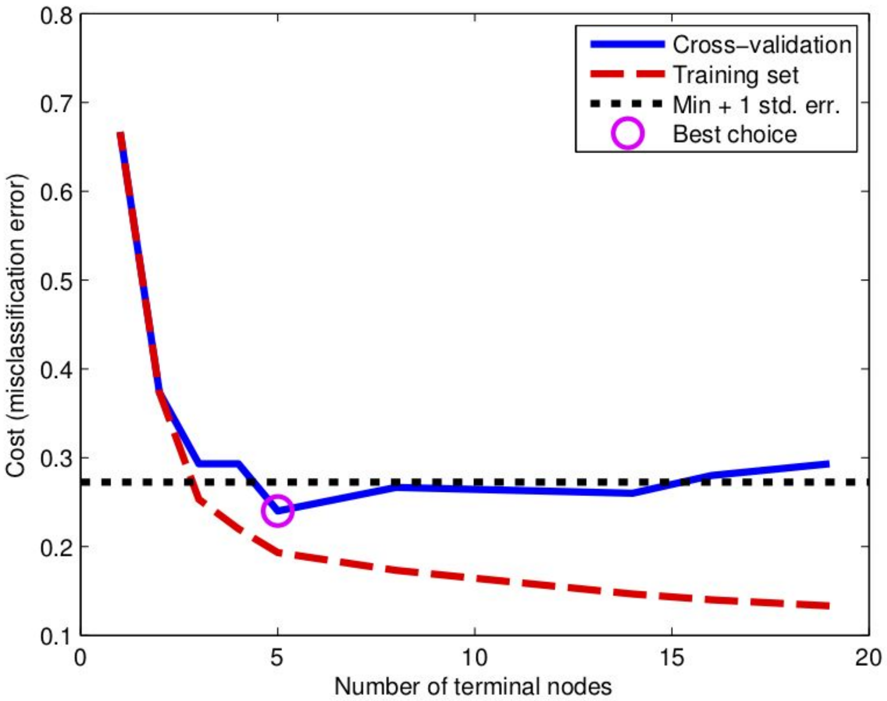
Decision Trees - Iris example (1)





Decision Trees - Iris example (2)





Bagging (**B**ootstrap **Ag**gregation)

Limiting depth isn't very precise, can quickly go from underfitting to overfitting

What if we could **smooth** the output somehow?

Bagging: Fit multiple "different" trees to the training data. For prediction return the class that the most trees returned.

$$\mathcal{H} = \left\{ h_E : h_E(\mathbf{x}) = \arg\max_c \sum_{h_k \in E} \mathbb{I}(h_k(\mathbf{x}) = c) \right\}$$

Random Forests

How do we get “different” trees if all we have is the one training dataset?

Idea 1: Randomly sample the data **with replacement** (the bootstrap part)

$$B_k = \left\{ (\mathbf{x}^{(i)}, y^{(i)})_k \right\}_{k=1}^{K}$$

$$i \sim U(\{1, \ldots, N\}) = p(i) = \frac{1}{N}$$

Idea 2: Pick split points and values **randomly** instead of optimally.

Component of x to split on

Uniform distribution

$$j^* \sim U(\{1, \ldots, D\}) = p(j) = \frac{1}{D}$$

$$t^* \sim U(\mathcal{T}_j) = p(t) = \frac{1}{|\mathcal{T}_j|}$$

“Question”/threshold to use

Set of possible thresholds

Random Forest algorithm

Loop k=1…K times:

1. Sample the training dataset M times, **with replacement**
2. Fit a “Random Tree” to the sampled data, call it h_k
3. Store the new tree

To classify a new point, **x**, get the output of $\hat{y}_k = h_k(\mathbf{x})$ for k=1…K, and

- For classification, return mode($\{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_K\}$)
- For regression, return average($\{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_K\}$)

“Random Tree” algorithm

Since it is OK for our individual trees to overfit, typically we do not use a max_depth parameter: tree continues to grow until it fits the training data exactly

random-tree-learning(examples, questions, default_val):

1. IF examples is empty THEN RETURN leaf(default_val)
2. ELSE IF all examples have same label THEN RETURN leaf(label)
3. ELSE IF remaining questions is empty THEN RETURN leaf(majority-label(examples))

   ELSE

   rand_q = question chosen at random

   node = new DT node with question rand_q

   subtree_default = majority-label(examples)

   subtree_questions = questions without rand_q

   FOREACH “v” response to rand_q DO:

   subset = {element of examples where rand_q(example)=v}

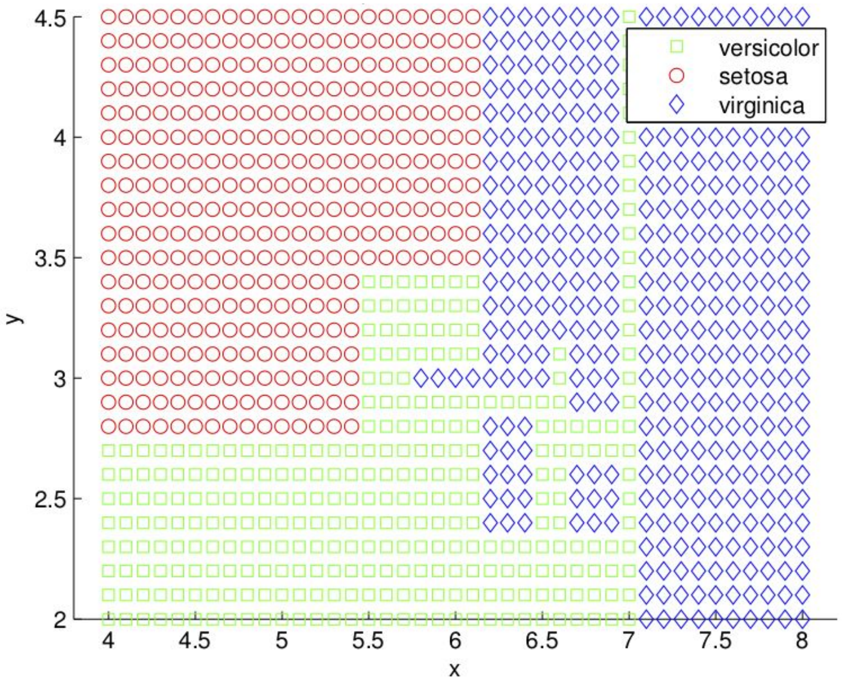
   recursion → subtree = random-tree-learning(subset,subtree_questions,subtree_default)

   add branch to node for v pointing to subtree

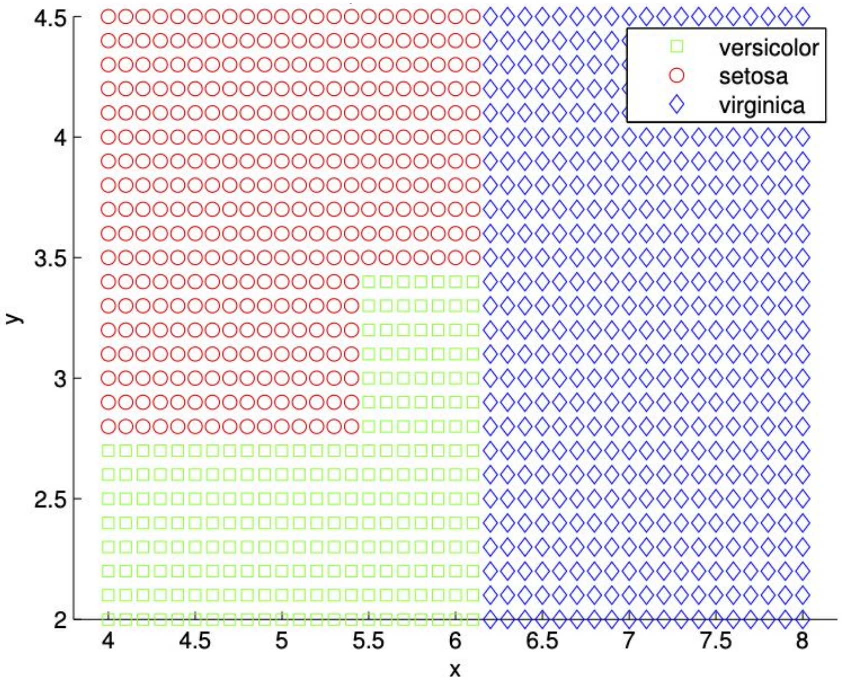
   return node

Comparing DTs and RFs (1)

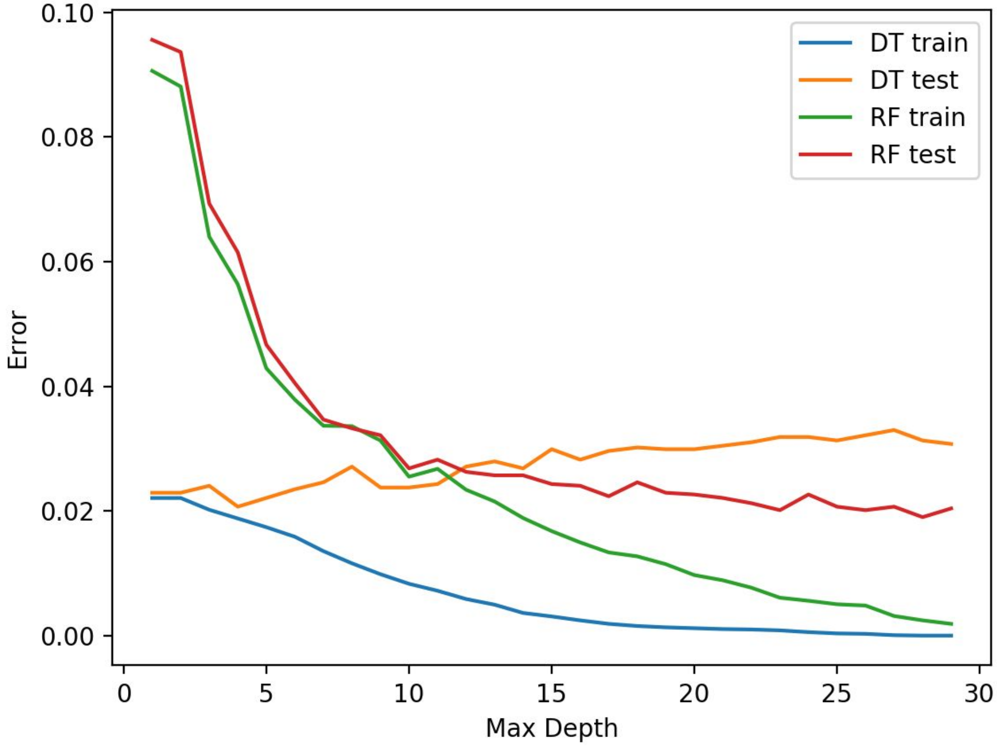
Single Decision Tree



Random Forest



Comparing DTs and RFs (2)



Boosting

Bagging can be memory/storage intensive, depending on how the data are distributed, and how the tree is represented.

Instead of starting with a very complex model and smoothing, combine the output of a bunch of very simple models to **boost** their performance.

Alternatively: an iterative process where each subsequent iteration tries to "correct" for the mistakes made by the previous set of learners. We can do this by **weighting** the training data that are misclassified.

AdaBoost

If the h_k are DT's, typically max_depth=1 or 2 (decision "stumps")

Start with uniform weights for all data points: $w_0^{(i)} = 1/N$

Trees with weighted data? Weight vote/avg at leaves.

Loop k=1…K times:

1. Fit h_k to the training data using weights $w_k^{(i)}$
2. Compute the weighted misclassification error
3. Compute new weights and a voting coefficient (β_k) for h_k

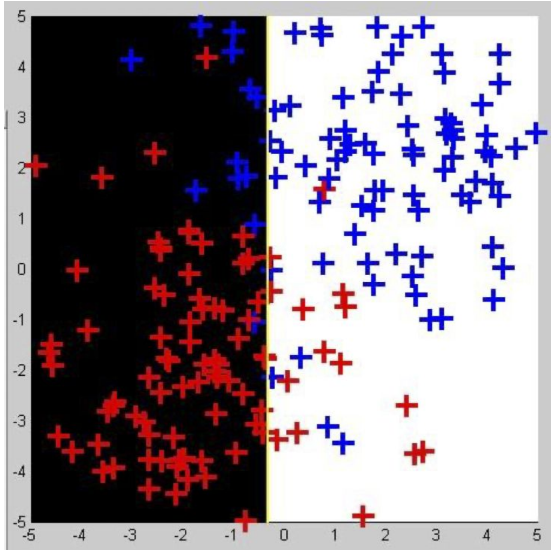
To classify a new point, **x**, compute the weighted vote from each h_k

$$\text{err}_k = \frac{\sum_{i=1}^{N} w^{(i)} \cdot \mathbb{I}\{h_k(\mathbf{x}^{(i)}) \neq y^{(i)}\}}{\sum_{i=1}^{N} w^{(i)}}$$

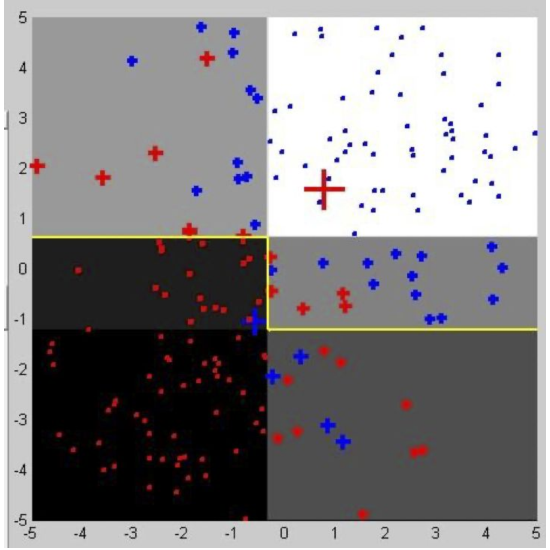
$$\beta_k = \log[(1 - \text{err}_k)/\text{err}_k]$$

$$w_k^{(i)} \leftarrow w_{k-1}^{(i)} \exp\left[\beta_k \cdot \mathbb{I}\{h_k(\mathbf{x}^{(i)}) \neq y^{(i)}\}\right]$$

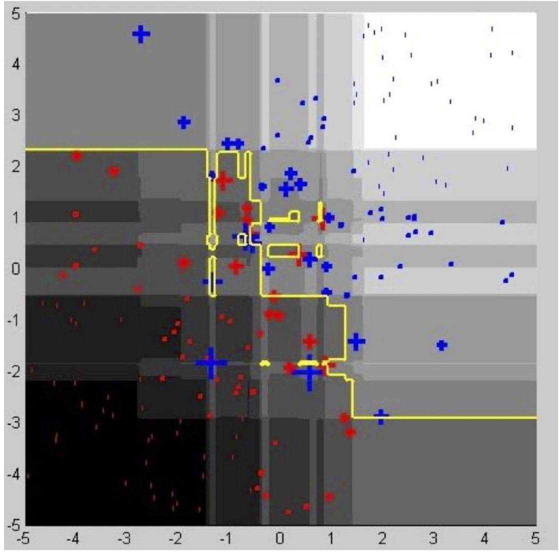
AdaBoost graphically



Initial decision stump



After 3 iterations



After 120 iterations

Boosting notes

- AdaBoost tends to be resistant to overfitting in practice
  - AdaBoost can be shown to maximize the "margin" between the decision boundary and the training data.
  - Under certain assumptions about noise, this makes AdaBoost extremely robust.
- Can be generalized beyond Decision Trees to work with any "weak learner"
  - A weak learner is one that does slightly better than randomly guessing
  - Using weighted data to train h_k can be tricky sometimes. One method that works with any learner: sample the training data according to the weight, then train as normal
- A version of this exists for regression instead of classification
  - Called **AdaBoost.R2**, needs a different definition for β_k and $w_k^{(i)}$ based on different loss

Summary

Wrapping Up

- Generalizations to DTs for continuous inputs and outputs
- Two different techniques for handling overfitting in DTs: Boosting and Bagging
- **Bagging**: train a collection of overfit trees and smooth the output
- **Boosting**: train a collection of underfit trees iteratively to improve performance on errors in the previous iteration
- Both boosting and bagging are examples of **ensemble** methods: techniques for combining the output of other ML algorithms