CSC2515 Winter 2015
Introduction to Machine Learning

Combining Models

All lecture slides will be available at:
http://www.cs.toronto.edu/~urtasun/courses/CSC2515/CSC2515_Winter15.html

Many of the figures are provided by Chris Bishop
from his textbook: ”Pattern Recognition and Machine Learning”

Ensemble methods

Typical application: classification

Ensemble of classifiers is a set of classifiers whose individual decisions combined in some way to classify new examples

Simplest approach:

1. Generate multiple classifiers
2. Each votes on test instance
3. Take majority as classification

Classifiers different due to different sampling of training data, or randomized parameters within the classification algorithm

Aim: take simple mediocre algorithm and transform it into a super classifier without requiring any fancy new algorithm

Ensemble methods: Summary

Differ in training strategy, and combination method

1. Parallel training with different training sets: bagging

2. Sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples: boosting

3. Parallel training with objective encouraging division of labor: mixture of experts

Notes:

- Also known as meta-learning
- Typically applied to weak models, such as decision stumps (single-node decision trees), or linear classifiers

Variance-bias tradeoff?

Minimize two sets of errors:

1. Variance: error from sensitivity to small fluctuations in the training set

2. Bias: erroneous assumptions in the model

Variance-bias decomposition is a way of analyzing the generalization error as a sum of 3 terms: variance, bias and irreducible error (resulting from the problem itself)

Why do ensemble methods work?

Based on one of two basic observations:

1. Variance reduction: if the training sets are completely independent, it will always helps to average an ensemble because this will reduce variance without affecting bias (e.g., bagging) -- reduce sensitivity to individual data pts

2. Bias reduction: for simple models, average of models has much greater capacity than single model (e.g., hyperplane classifiers, Gaussian densities). Averaging models can reduce bias substantially by increasing capacity, and control variance by fitting one component at a time (e.g., boosting)

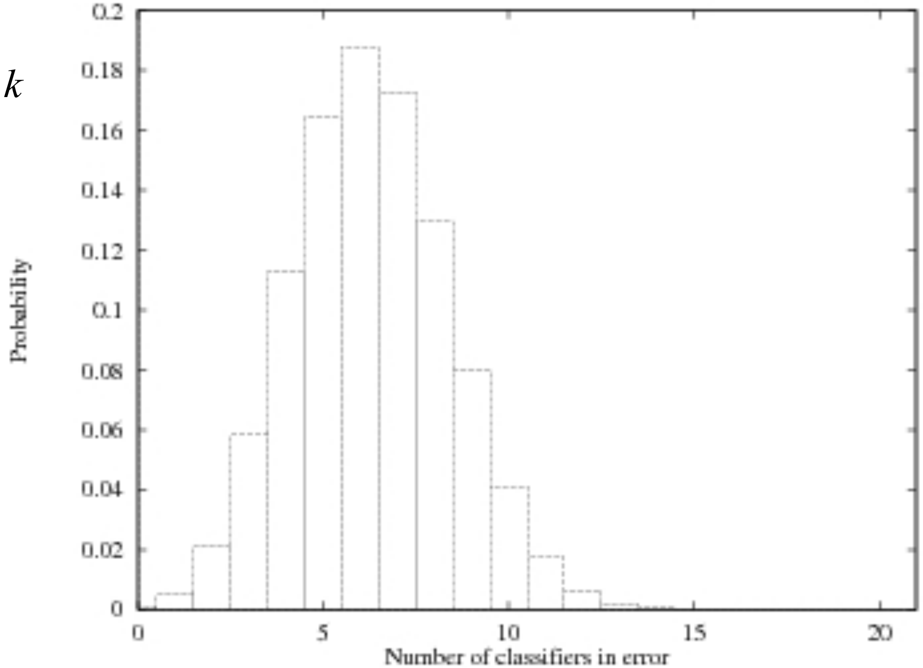
Ensemble methods: Justification

Ensemble methods more accurate than any individual members:

- Accurate (better than guessing)
- Diverse (different errors on new examples)

Independent errors: prob *k* of *N* classifiers (independent error rate ε) wrong:

$$P(\#\text{errrors} = k) = \binom{N}{k} \varepsilon^{k} (1-\varepsilon)^{N-k}$$



Probability that majority vote wrong: error under distribution where more than *N/2* wrong

Ensemble methods: Netflix

Clear demonstration of the power of ensemble methods

Original progress prize winner (BellKor) was ensemble of 107 models!

> *"Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a simple technique."*

> *"We strongly believe that the success of an ensemble approach depends on the ability of its various predictors to expose different complementing aspects of the data. Experience shows that this is very different than optimizing the accuracy of each individual predictor."*

Bootstrap estimation

- Repeatedly draw *n* samples from *D*
- For each set of samples, estimate a statistic
- The bootstrap estimate is the mean of the individual estimates
- Used to estimate a statistic (parameter) and its variance
- Bagging: bootstrap aggregation (Breiman 1994)

Bagging

Simple idea: generate M bootstrap samples from your original training set. Train on each one to get y_m, and average them

$$y_{bag}^{M}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x})$$

For regression: average predictions

For classification: average class probabilities (or take the majority vote if only hard outputs available)

Bagging approximates the Bayesian posterior mean. The more bootstraps the better, so use as many as you have time for

Each bootstrap sample is drawn *with replacement,* so each one contains some duplicates of certain training points and leaves out other training points completely

Boosting (AdaBoost): Summary

Also works by manipulating training set, but classifiers trained sequentially

Each classifier trained given knowledge of the performance of previously trained classifiers: focus on hard examples

Final classifier: weighted sum of component classifiers

Making weak learners stronger

- Suppose you have a weak learning module (a “base classifier”) that can always get 0.5+epsilon correct when given a two-way classification task
  - That seems like a weak assumption but beware!
- Can you apply this learning module many times to get a strong learner that can get close to zero error rate on the training data?
  - Theorists showed how to do this and it actually led to an effective new learning procedure (Freund & Shapire, 1996)

Boosting (ADAboost)

- First train the base classifier on all the training data with equal importance weights on each case.
- Then re-weight the training data to emphasize the hard cases and train a second model.
  - How do we re-weight the data?
- Keep training new models on re-weighted data
- Finally, use a weighted committee of all the models for the test data.
  - How do we weight the models in the committee?

How to train each classifier

$$\text{input}: \mathbf{x}, \quad \text{output}: \; y(\mathbf{x}) \in \{1,-1\}$$

$$\text{target}: \quad t \in \{1,-1\},$$

$$\text{weight on case } n \text{ for classifier } m: \quad w_n^m$$

$$\text{Cost function for classifier } m:$$

$$J_m = \sum_{n=1}^{N} w_n^m \left[y_m(\mathbf{x}_n) \neq t_n \right] = \sum \textit{weighted errors}$$

1 if error,
0 if correct

How to weight each training case for classifier m

$$Let \quad \varepsilon_m = \frac{J_m}{\sum_n w_n^m}$$

← weighted error rate of classifier

$$Let \quad \alpha_m = \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$$

← This is the quality of the classifier. It is zero if the classifier has weighted error rate of 0.5 and infinity if the classifier is perfect

$$w_n^{m+1} = w_n^m \exp\{ \alpha_m [y_m(x_n) \neq t_n] \}$$

How to make predictions using a committee of classifiers

- Weight the binary prediction of each classifier by the quality of that classifier:

$$Y_M(\mathbf{x}) \quad = \quad sign\left(\sum_{m=1}^{M} \alpha_m \, y_m(\mathbf{x}) \right)$$

An alternative derivation of ADAboost

Just write down the right cost function and optimize each parameter to minimize it -- stagewise additive modeling (Friedman et. al. 2000)

$$E = \sum_{n=1}^{N} \exp\left\{ -t_n \, f_m(\mathbf{x}_n) \right\}$$

← the exponential loss for classifier m

$$f_m(\mathbf{x}) = \frac{1}{2} \sum_{l=1}^{l=m} \alpha_l \, y_l(\mathbf{x})$$

← real-valued prediction by committee of models up to m

Learning classifier m using exponential loss

$$E = \sum_{n=1}^{N} \exp\left\{-t_n\, f_m(\mathbf{x}_n)\right\}$$

$$f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{l=m} \alpha_l y_l(\mathbf{x}) = \frac{1}{2}\alpha_m\, y_m(\mathbf{x}) \;+\; \frac{1}{2}\sum_{l=1}^{l=m-1} \alpha_l y_l(\mathbf{x})$$

$$E_{relevant} = \sum_{n=1}^{N} \exp\left\{-t_n\, f_{m-1}(\mathbf{x}_n) - \frac{1}{2} t_n\, \alpha_m\, y_m(\mathbf{x}_n)\right\}$$

$$= \sum_{n=1}^{N} w_n^m \exp\left\{-\frac{1}{2} t_n\, \alpha_m\, y_m(\mathbf{x}_n)\right\}$$

Re-writing the part of the exponential loss that is relevant when fitting classifier m

$$E_{relevant} = \sum_{n=1}^{N} w_n^m \exp\left\{ - t_n \frac{\alpha_m}{2} y_m(\mathbf{x}_n) \right\}$$

$$= e^{-\frac{\alpha_m}{2}} \sum_{right} w_n^m \quad + \quad e^{+\frac{\alpha_m}{2}} \sum_{wrong} w_n^m$$

unmodifiable

$$= \left(e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}} \right) \sum_n w_n^m \left[t_n \neq y_m(\mathbf{x}_n) \right] \quad + \quad e^{-\frac{\alpha_m}{2}} \sum_n w_n^m$$

multiplicative constant

wrong cases

AdaBoost algorithm

Input:

- Training set examples $\{\boldsymbol{x}_n, t_n\}$, $n = 1,2,\ldots,N$; $t_n \in \{-1,+1\}$
- *WeakLearn*: learning procedure, produces classifier $y(\boldsymbol{x})$

Initialize example weights: $w_n^m(\boldsymbol{x}) = 1/N$

For m=1:M

- $y_m(\boldsymbol{x}) = WeakLearn(\{\boldsymbol{x}\},\boldsymbol{t},\boldsymbol{w})$ – fit classifier to training data, minimizing weighted error function:

$$J_m = \sum_{n=1}^{N} w_n^m [y_m(\mathbf{x}_n) \neq t_n]$$

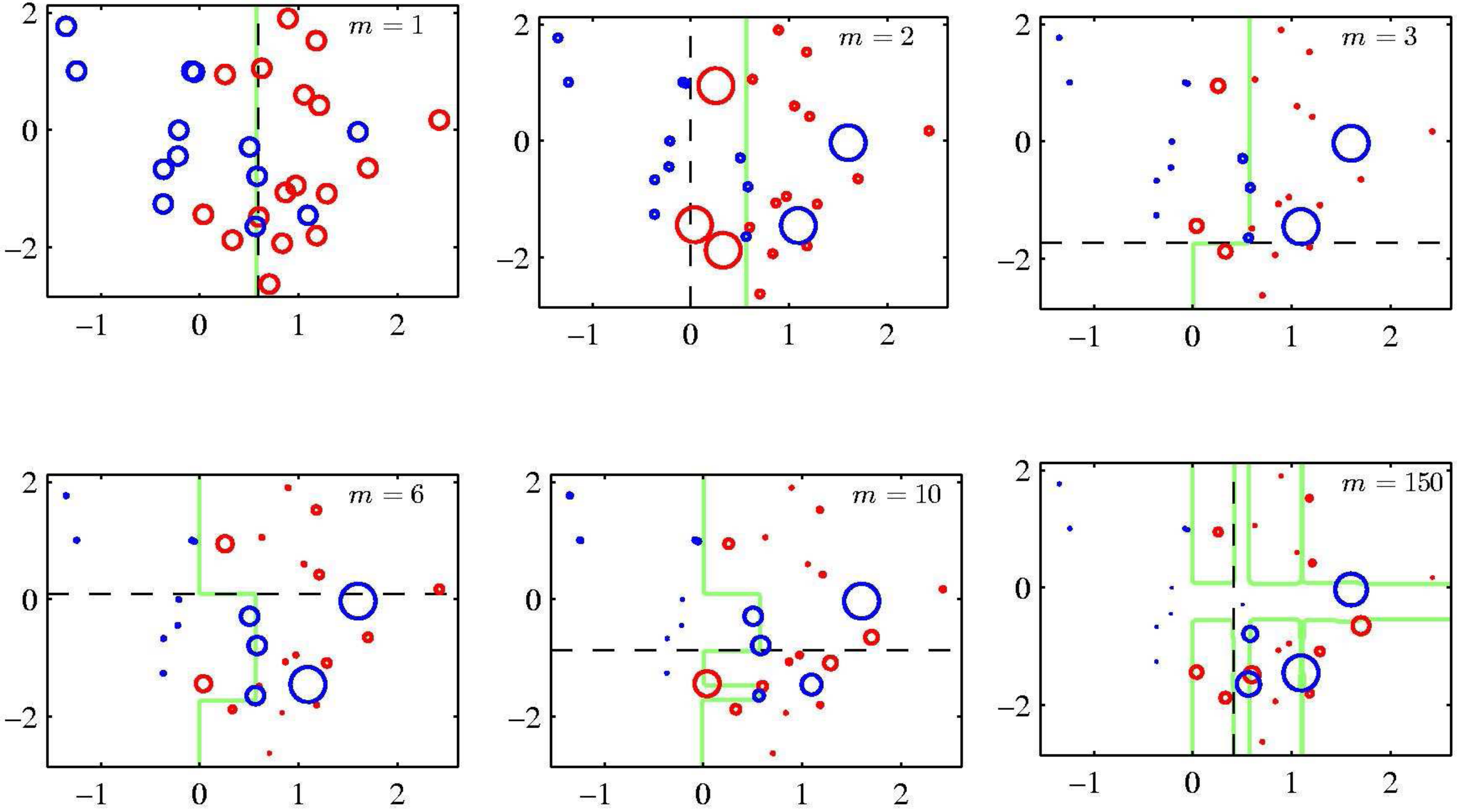
- error rate: $\varepsilon_m = \sum_{n=1}^{N} w_n^m [y_m(\mathbf{x}_n) \neq t_n]$
- classifier coefficient: $\alpha_m = \frac{1}{2}\log\{(1-\varepsilon_m)/\varepsilon_m\}$
- update data weights: $w_n^{m+1} = w_n^m \exp\{-\alpha_m t_n y_m(\mathbf{x}_n)\}/Z^{m+1}$

$$Z^{m+1} = \sum_{n=1}^{N} w_n^m \exp\{-\alpha_m t_n y_m(\mathbf{x}_n)\}$$

Final model:

$$Y^M(\mathbf{x}) = \mathrm{sign}(y^M(\mathbf{x})) = \mathrm{sign}(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}))$$

AdaBoost example



An impressive example of boosting

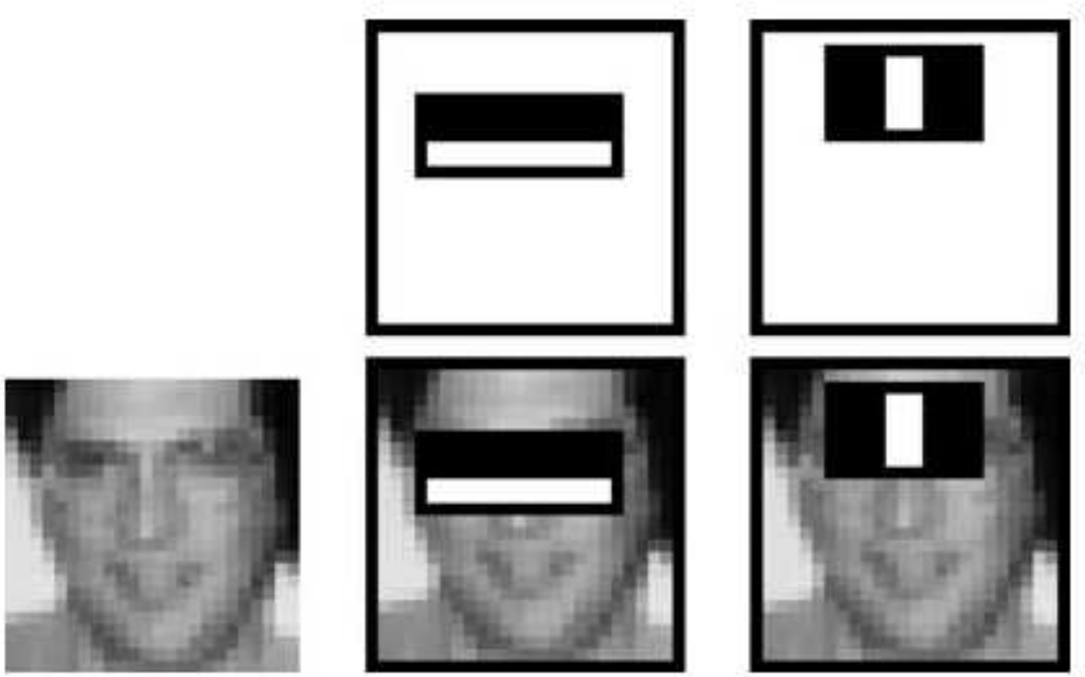
- Viola and Jones created a very fast face detector that can be scanned across a large image to find the faces.
- The base classifier/weak learner just compares the total intensity in two rectangular pieces of the image.
  - There is a neat trick for computing the total intensity in a rectangle in a few operations.
    - So its easy to evaluate a huge number of base classifiers and they are very fast at runtime.
  - The algorithm adds classifiers greedily based on their quality on the weighted training cases.

AdaBoost in face detection

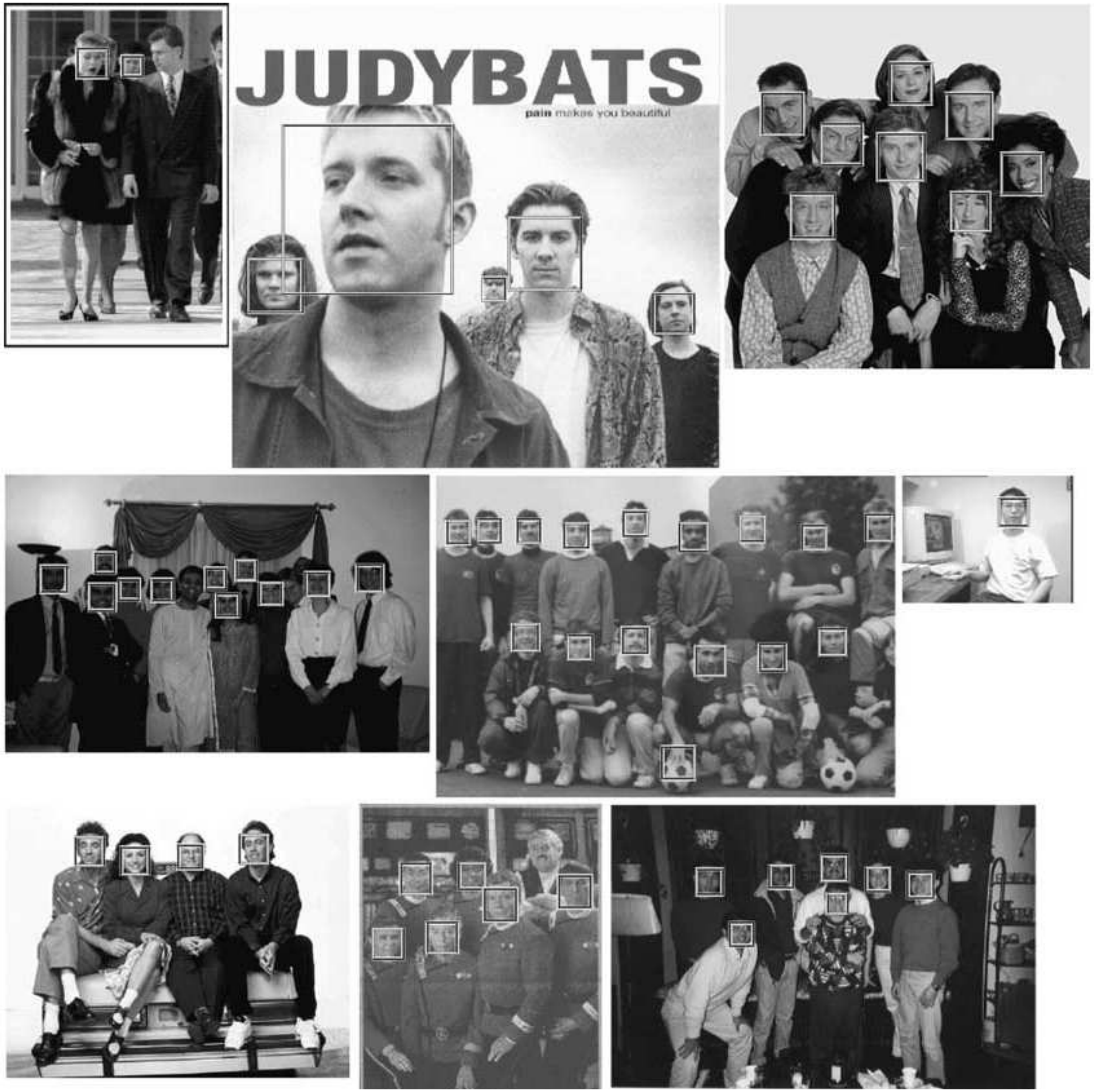
Famous application of boosting: detecting faces in images

Two twists on standard algorithm

1) Pre-define weak classifiers, so optimization=selection
2) Change loss function for weak learners: false positives less costly than misses



AdaBoost face detection results



AdaBoost: Maximizing margin?

Many attempts to explain AdaBoost's tendency to not overfit – even though apparent complexity of classifier grows each round, test error tends not to go up

$$\text{margin}(\mathbf{x}_n) = t_n \sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}_n) / \sum_{m=1}^{M} \alpha_m$$

- Magnitude: strength of agreement of classifiers
- Sign: weight combination produces correct prediction

Bound on generalization error not dependent on number of rounds

$$\Pr[y_m(\mathbf{x}_n) \neq t_n] \leq \Pr[\text{margin}(x) \leq \theta] + O\left(\sqrt{\frac{d}{N\theta^2}}\right)$$

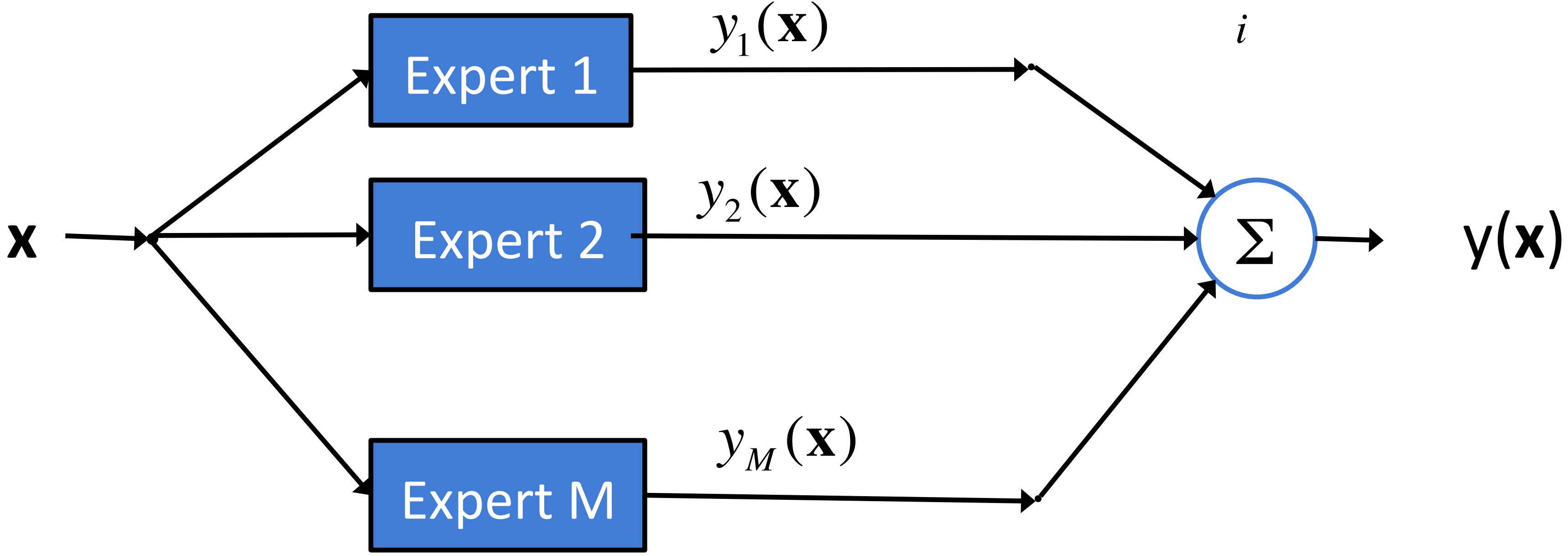
Depends on minimum margin: other algorithms directly maximizes this

Appears to produce higher margins distribution yet worse test error

Ensemble learning: Boosting and Bagging

- Experts cooperate to predict output

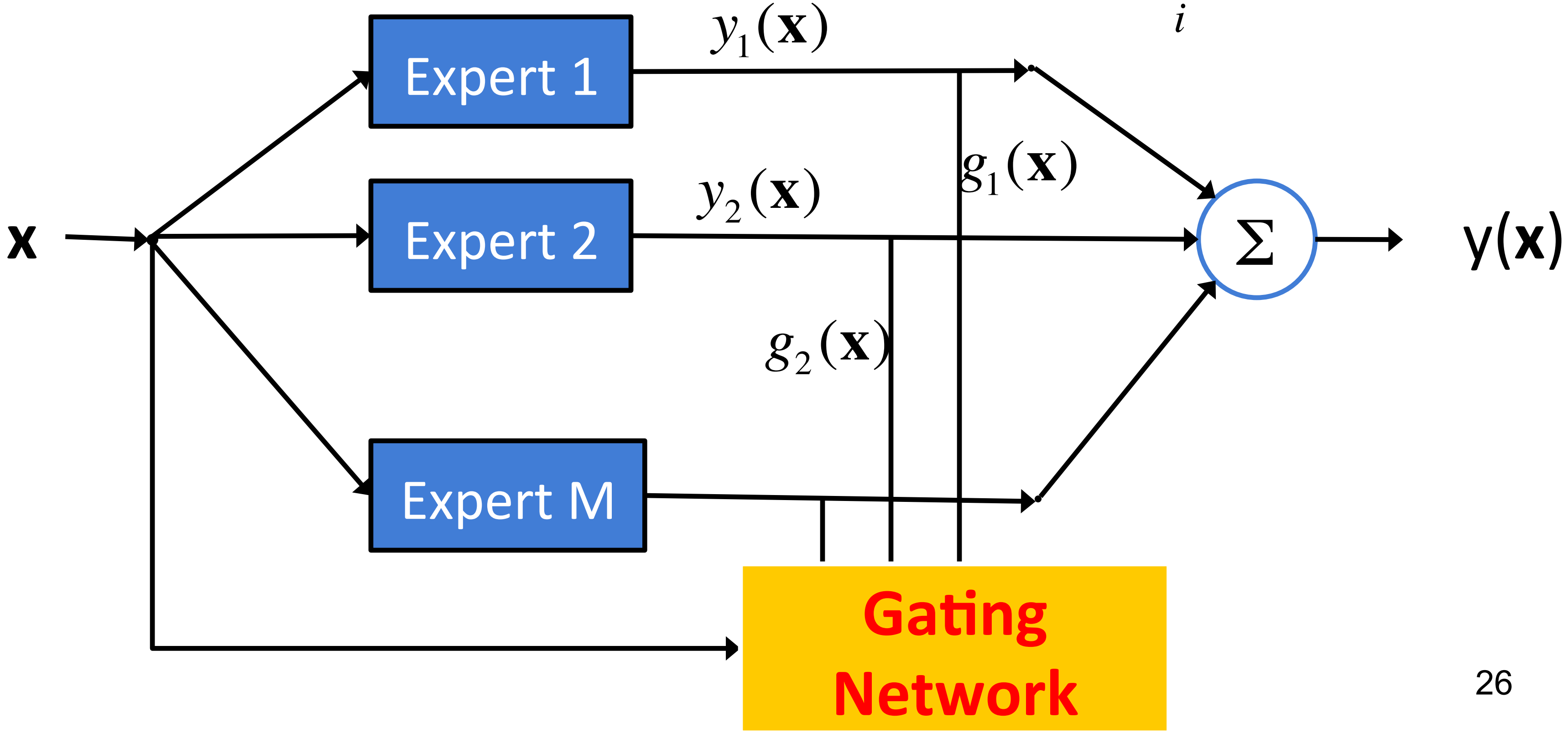
$$y(\mathbf{x}) = \sum_i g_i y_i(\mathbf{x})$$



Mixture of Experts

Gating network encourages specialization (local experts) instead of cooperation

$$y(\mathbf{x}) = \sum_i g_i(\mathbf{x}) y_i(\mathbf{x})$$



Mixture of Experts: Summary

1. Cost function designed to make each expert estimate desired output independently

2. Gating network softmax over experts: stochastic selection of who is the true expert for given input

3. Allow each expert to produce distribution over outputs

Cooperation vs. Specialization

- Consider regression problem

- To encourage cooperation, can train to reduce discrepancy between average of predictors with target

$$E = (t - \frac{1}{M}\sum_m y_m)^2$$

- This can overfit badly. It makes the model much more powerful than training each predictor separately.

- Leads to odd objective: consider adding models/experts sequentially – if its estimate for t is too low, and the average of other models is too high, then model m encouraged to *lower* its prediction

Cooperation vs. Specialization

- To encourage specialization, train to reduce the average of each predictor's discrepancy with target

$$E = \frac{1}{M}\sum_m (t - y_m)^2$$

- use a weighted average: weights are probabilities of picking that "expert" for the particular training case.

$$E = \frac{1}{M}\sum_m g_m(\mathbf{x})(t - y_m(\mathbf{x}))^2$$

- Gating output is softmax of z = Ux

$$g_m(\mathbf{x}) = \exp(z_m(\mathbf{x})) / \sum_{m'} \exp(z_{m'}(\mathbf{x}))$$

Derivatives of simple cost function

- Look at derivatives to see what cost function will do

$$E = \frac{1}{M}\sum_m g_m(\mathbf{x})(t - y_m(\mathbf{x}))^2$$

- For gating network, increase weight on expert when its error less than average error of experts

$$\frac{\partial E}{\partial y_m} = \frac{1}{M} g_m(\mathbf{x})(t - y_m(\mathbf{x}))$$

$$\frac{\partial E}{\partial z_m} = \frac{1}{M} g_m(\mathbf{x})[(t - y_m(\mathbf{x}))^2 - E]$$

Mixture of Experts: Final cost function

- Can improve cost function by allowing each expert to produce not just single value estimate, but distribution

- Result is a mixture model

$$p(y \mid MOE) = \sum_m g_m(\mathbf{x}) \mathbf{N}(y \mid y_m(\mathbf{x}), \Sigma)$$

$$-\log p(t \mid MOE) = -\log \sum_m g_m(\mathbf{x}) \exp(-\| t - y_m(\mathbf{x}) \|^2 /2)$$

- Gradient: Error weighted by posterior probability of the expert

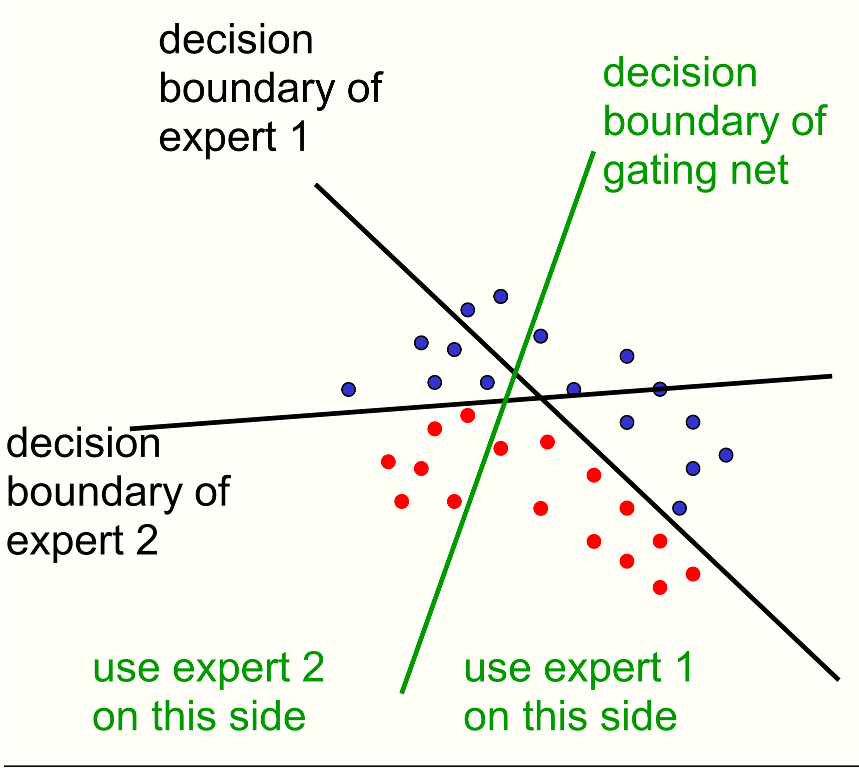
$$\frac{\partial E}{\partial y_m} = -2 \frac{g_m(\mathbf{x}) \exp(-\| t - y_m(\mathbf{x}) \|^2 /2)}{\sum_{m'} g_{m'}(\mathbf{x}) \exp(-\| t - y_{m'}(\mathbf{x}) \|^2 /2)} (t - y_m(\mathbf{x}))$$

Mixture of Experts: Summary

1. Cost function designed to make each expert estimate desired output independently

2. Gating network softmax over experts: stochastic selection of who is the true expert for given input

3. Allow each expert to produce distribution over outputs


decision boundary of expert 1
decision boundary of gating net
decision boundary of expert 2
use expert 2 on this side
use expert 1 on this side

Ensemble methods: Summary

Differ in training strategy, and combination method

- Parallel training with different training sets
  1. Bagging (bootstrap aggregation) – train separate models on overlapping training sets, average their predictions
  2. Cross-validated committees – disjoint subsets of training sets
- Sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples: boosting
- Parallel training with objective encouraging division of labor: mixture of experts

Notes:

- Differ in: training strategy; selection of examples; weighting of components in final classifier

What are the base classifiers?

Popular choices of base classifier for boosting and other ensemble methods:

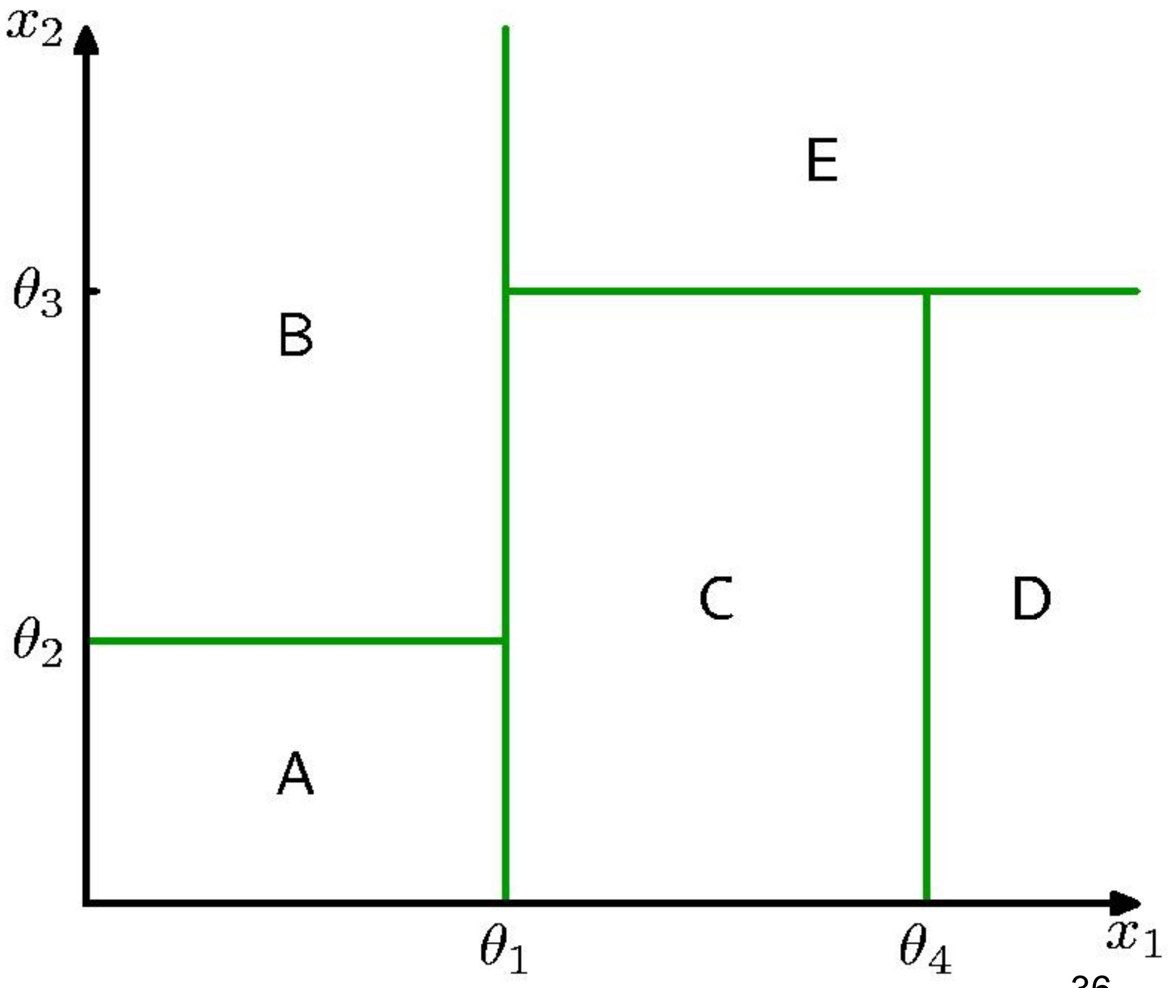
- Linear classifiers
- Decision trees

Decision Trees: Non-linear regression or classification with very little computation

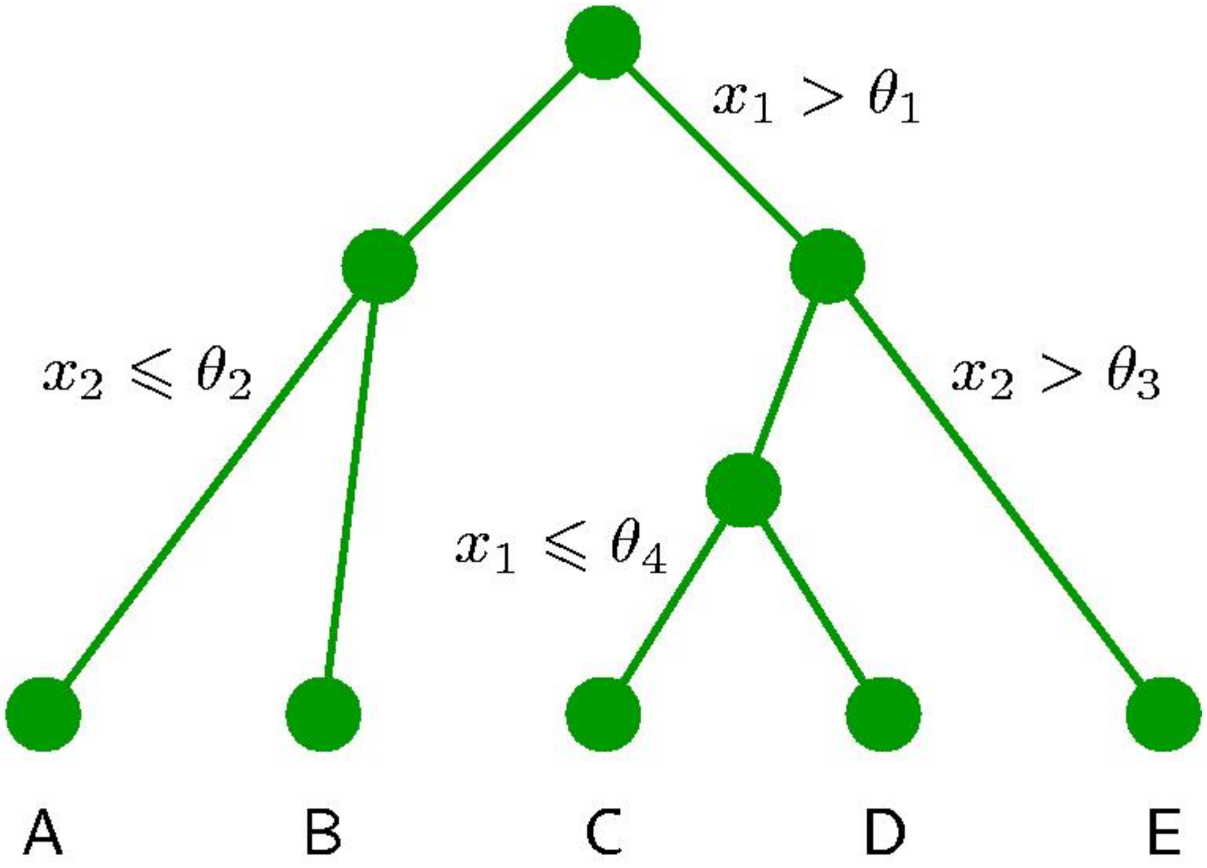
- The idea is to divide up the input space into a disjoint set of regions and to use a very simple estimator of the output for each region
  - For regression, the predicted output is just the mean of the training data in that region.
    - But we could fit a linear function in each region
  - For classification the predicted class is just the most frequent class in the training data in that region.
    - We could estimate class probabilities by the frequencies in the training data in that region.

A very fast way to decide if a datapoint lies in a region

- We make the decision boundaries orthogonal to one axis of the space and parallel to all the other axes.
    - This is easy to illustrate in a 2-D space
- Then we can locate the region that a datapoint lies in using a number of very simple tests that is logarithmic in the number of regions.



An axis-aligned decision tree

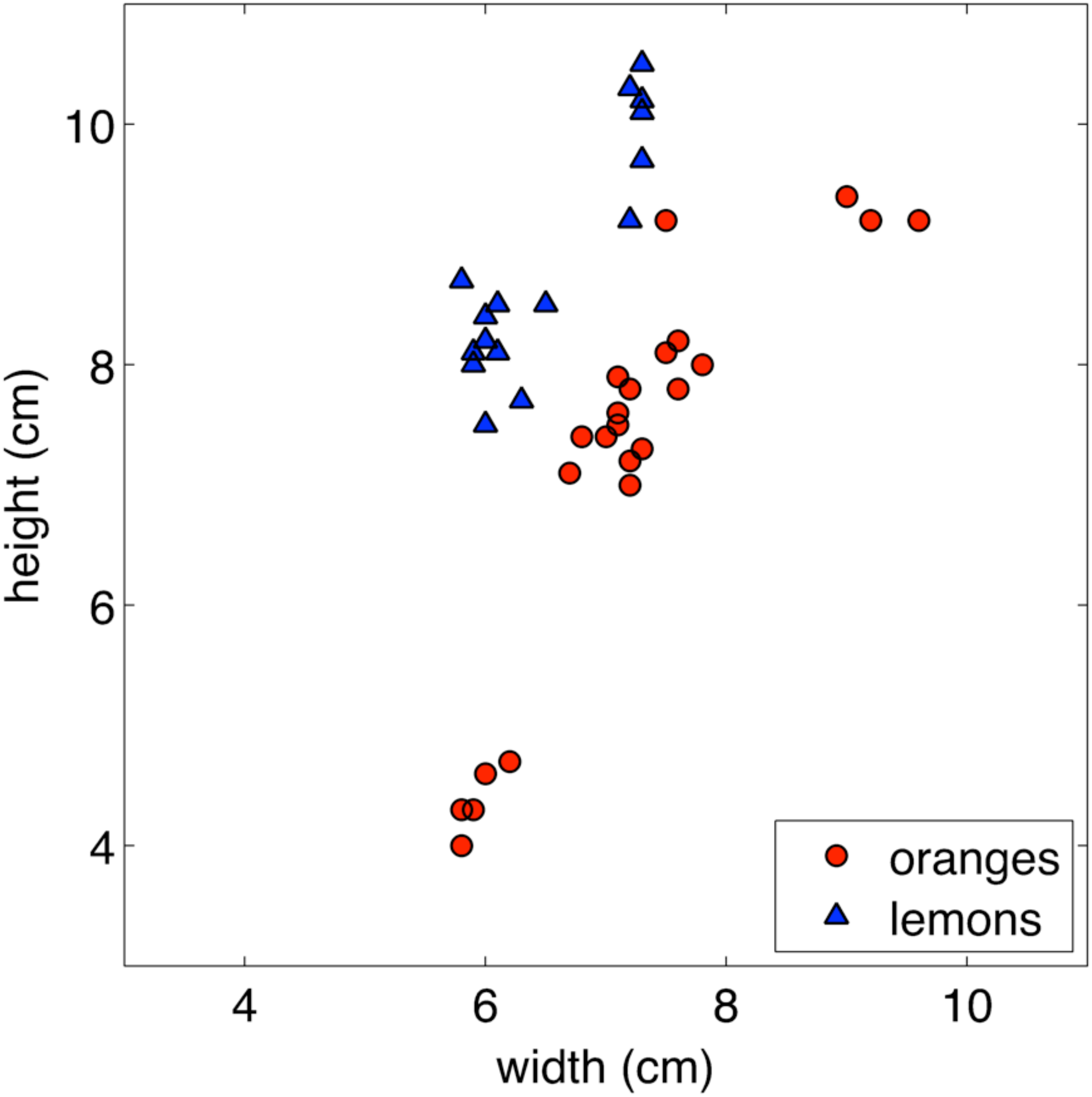


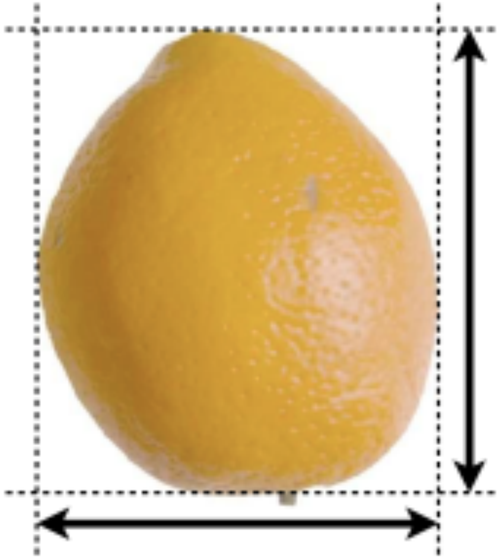
Decision Trees

- Internal nodes **test attributes**.
- Branching is determined by **attribute value**.
- Leaf nodes are **outputs** (class assignments).

In general, a decision tree can represent any binary function.

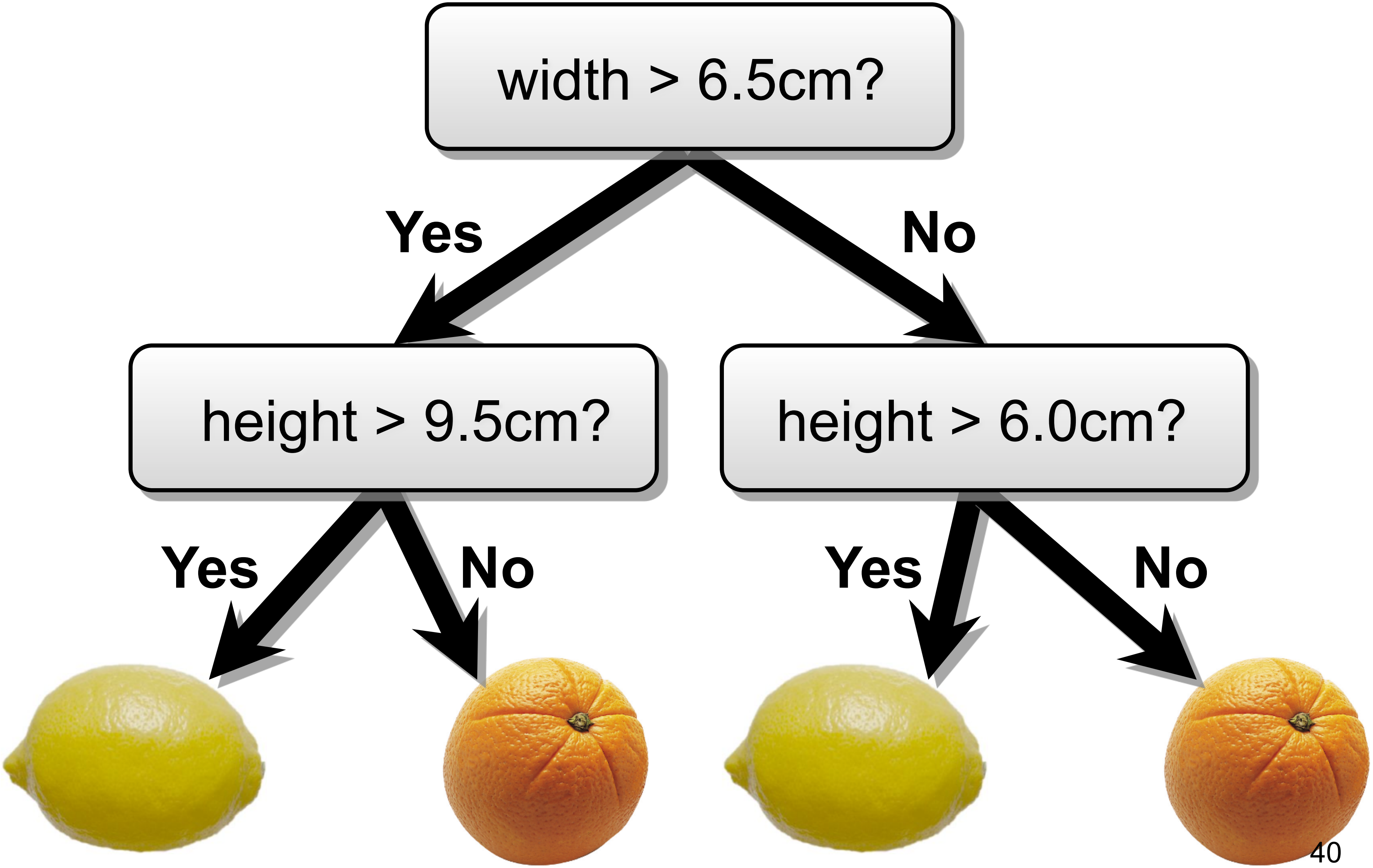
Example: Classifying Oranges and Lemons





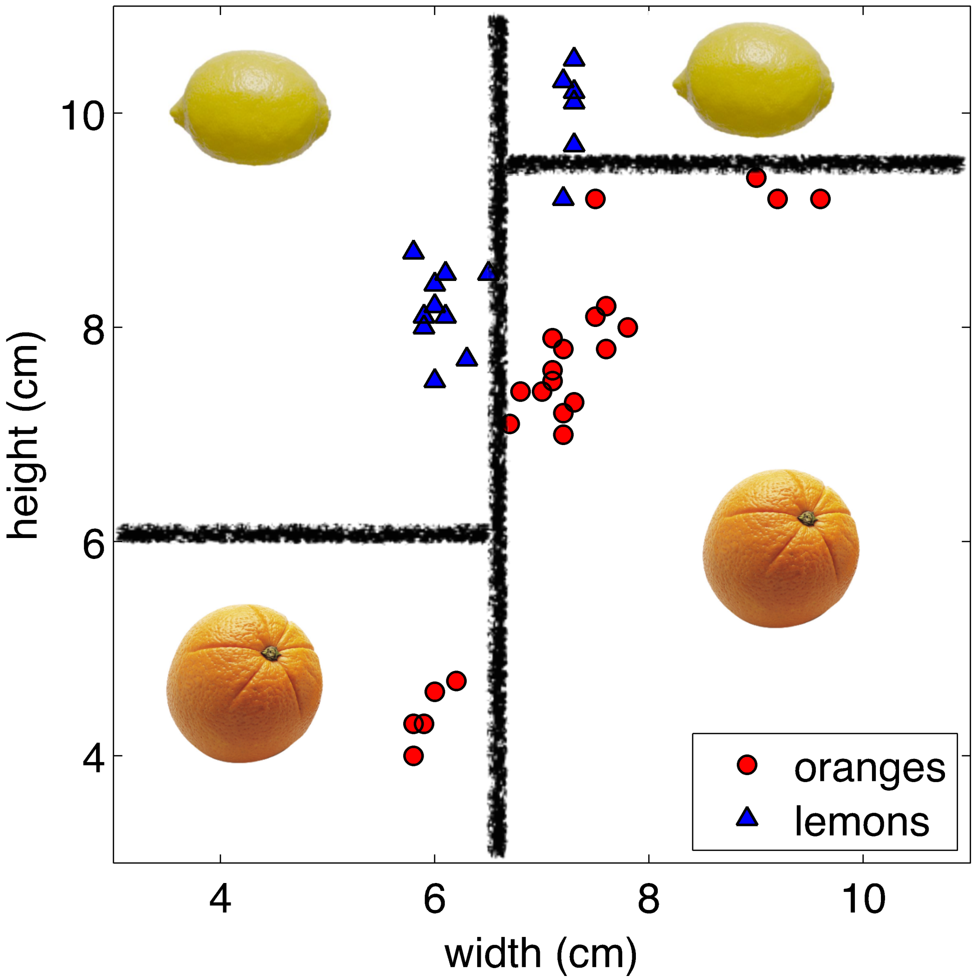
Data from Iain Murray

Decision Trees



Decision Tree: Decision Boundary

We could view the “decision boundary” as being the composition of several simple boundaries.



Decision Trees: Outline

- Choose a category in which to descend at each level.
- Condition on earlier (higher) choices.
- Generally, restrict only one dimension at a time.
- Declare an output value when you get to the bottom.

In the orange/lemon example, we only split each dimension once, but that is not required.

How do you construct a useful decision tree?

Constructing Decision Trees

Many variations (CART, ID3, C4.5) but share basic approach: recursively partition space by greedily picking the best test from a fixed set of possible tests at each step

At each level, one must choose:
1) Which variable to split.
2) Possibly where to split it.

- We need to decide on the set of possible tests
  - Consider splits at every coordinate of every point in the training data (i.e. all axis-aligned hyper-planes that touch datapoints)
- We need a measure of how good a test is ("purity")
  - Regression: resulting sum-squared error over all partitions.
  - For classification it is a bit more complicated.

We can use the **mutual information** to automate the process, and determine these selections.

Entropy of a Joint Distribution

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

$$
\begin{aligned}
H(X,Y) &= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(x,y) \\
&= -\frac{24}{100}\log_2\frac{24}{100} - \frac{1}{100}\log_2\frac{1}{100} - \frac{25}{100}\log_2\frac{25}{100} - \frac{50}{100}\log_2\frac{50}{100} \\
&\approx 1.56 \text{ bits}
\end{aligned}
$$

Specific Conditional Entropy

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

What is the entropy of cloudiness, given that it is raining?

$$\begin{aligned} H(X \mid Y = y) &= -\sum_{x \in X} p(x \mid y) \log_2 (x \mid y) \\ &= -\frac{24}{25} \log_2 \frac{24}{25} - \frac{1}{25} \log_2 \frac{1}{25} \\ &\approx 0.24 \text{ bits} \end{aligned}$$

(Non-Specific) Conditional Entropy

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

The expected conditional entropy:

$$\begin{aligned} H(X \mid Y) &= \sum_{y \in Y} p(y) H(X \mid Y = y) \\ &= -\sum_{y \in Y} p(y) \sum_{x \in X} p(x \mid y) \log_2(x \mid y) \end{aligned}$$

(Non-Specific) Conditional Entropy

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

What is the entropy of cloudiness, given the knowledge of whether or not it is raining?

$$\begin{aligned} H(X \mid Y) &= \sum_{y \in Y} p(y) H(X \mid Y = y) \\ &= \frac{1}{4} H(\text{clouds} \mid \text{is raining}) + \frac{3}{4} H(\text{clouds} \mid \text{not raining}) \\ &\approx 0.75 \text{ bits} \end{aligned}$$

Information Gain

- Treat frequencies of classes in each partition as probabilities and compute the entropy.
- Choose attribute based on how much information we would gain from the decision

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

How much information about cloudiness do we get by discovering whether it is raining?

$$\begin{aligned} IG(X \mid Y) &= H(X) - H(X \mid Y) \\ &\approx 0.25 \text{ bits} \end{aligned}$$

Also called **information gain** in X due to Y.

When should we stop adding nodes?

- Sometimes, the error stays constant for a while as nodes are added and then it falls
  - So we cannot stop adding nodes as soon as the error stops falling.
- It typically works best to fit a tree that is too large and then prune back the least useful nodes to balance complexity against error
  - We could use a validation set to do the pruning.

Advantages & disadvantages of decision trees

- They are easy to fit, easy to use, and easy to interpret as a fixed sequence of simple tests.

- They are non-linear, so they work much better than linear models for highly non-linear functions.

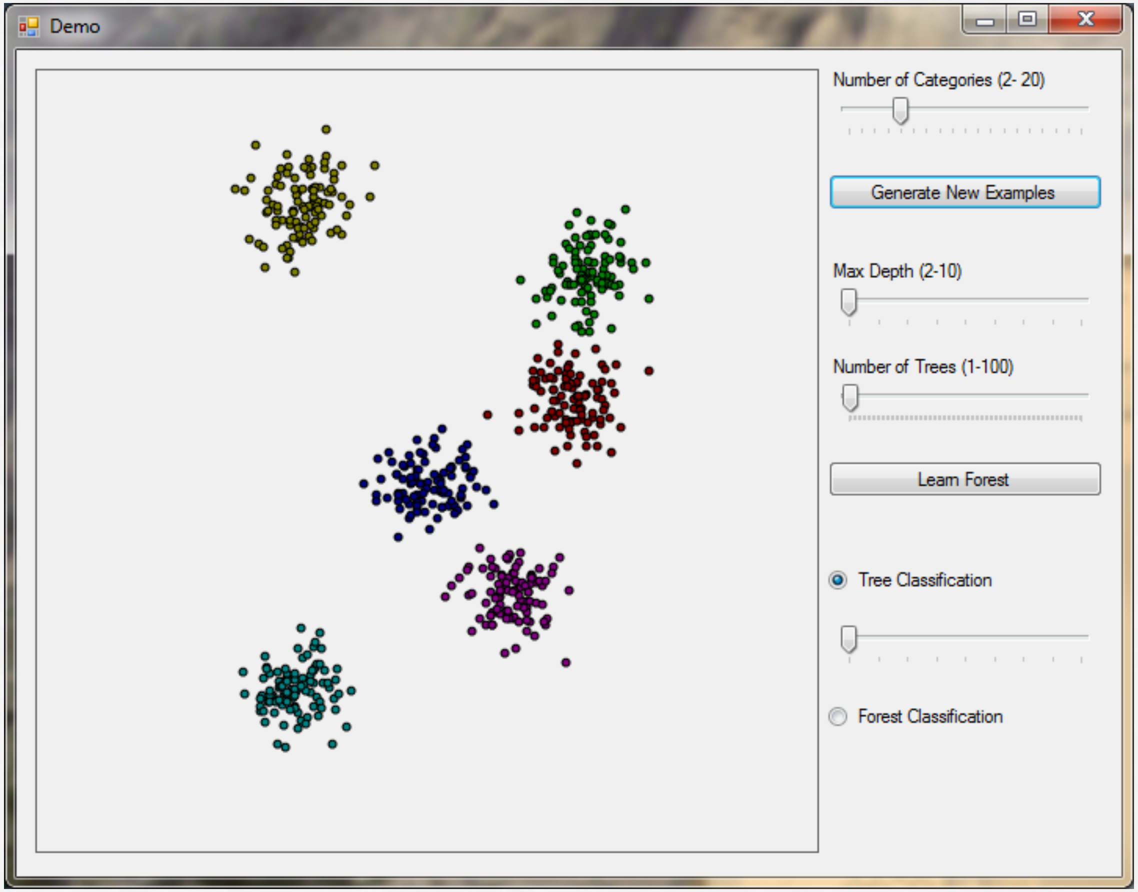
- They typically generalize less well than non-linear models that use adaptive basis functions, but its easy to improve them by averaging the predictions of many trees

Random/Decision Forests

- Definition: Ensemble of decision trees

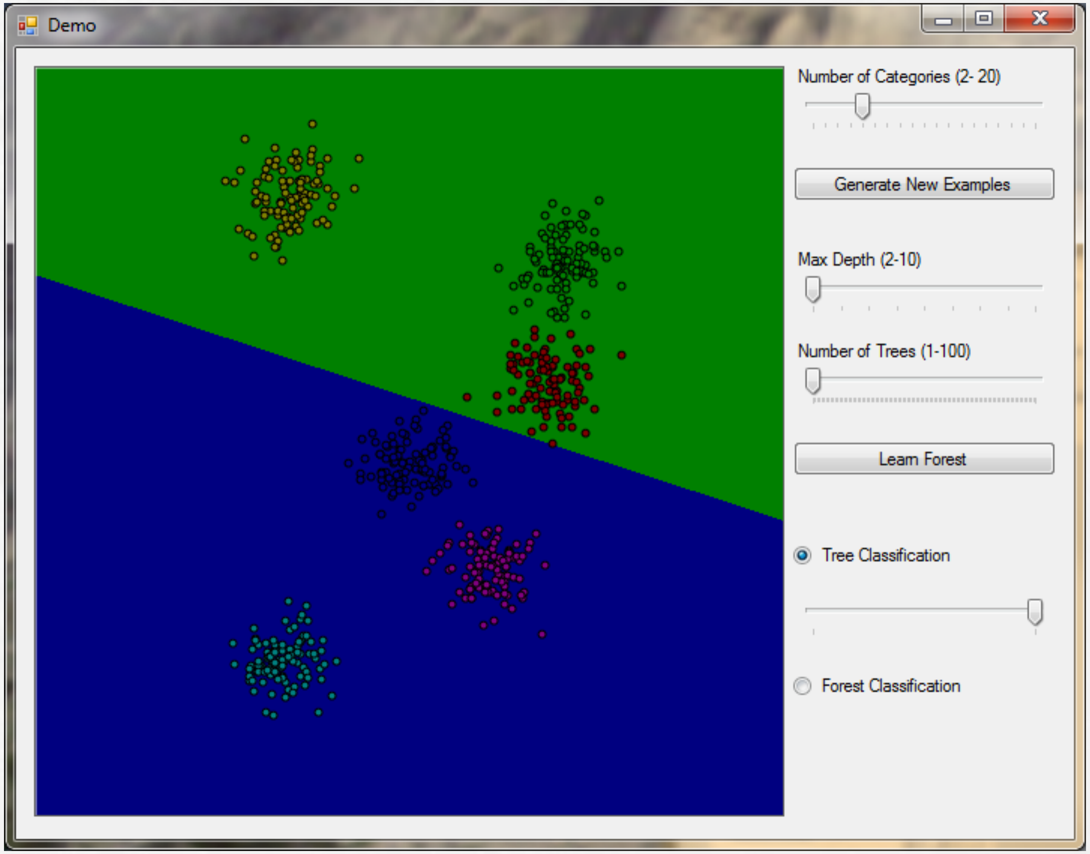
- Algorithm:
  - Divide training examples into multiple training sets (bagging)
  - Train a decision tree on each set (can randomly select subset of variables to consider)
  - Aggregate the predictions of each tree to make classification decision (e.g., can choose mode vote)

Toy Forest Classification Demo



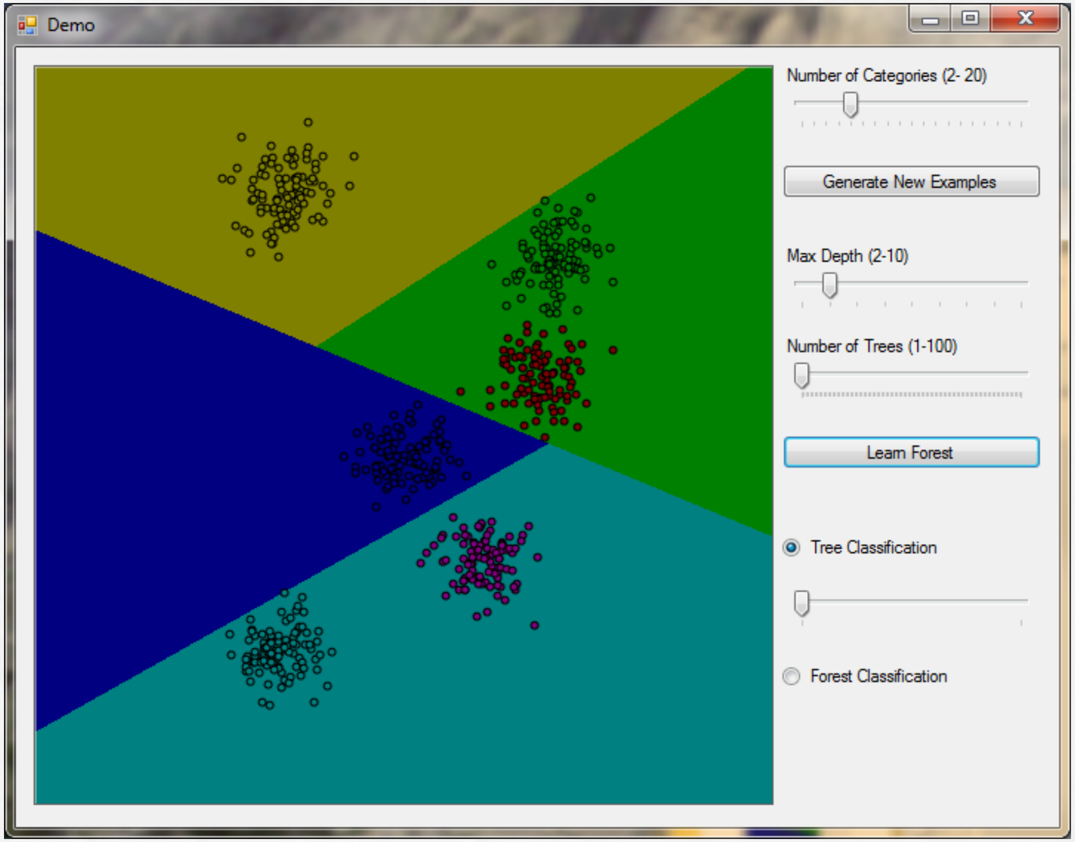
6 classes in a 2 dimensional feature space.
Split functions are lines in this space.

Toy Forest Classification Demo



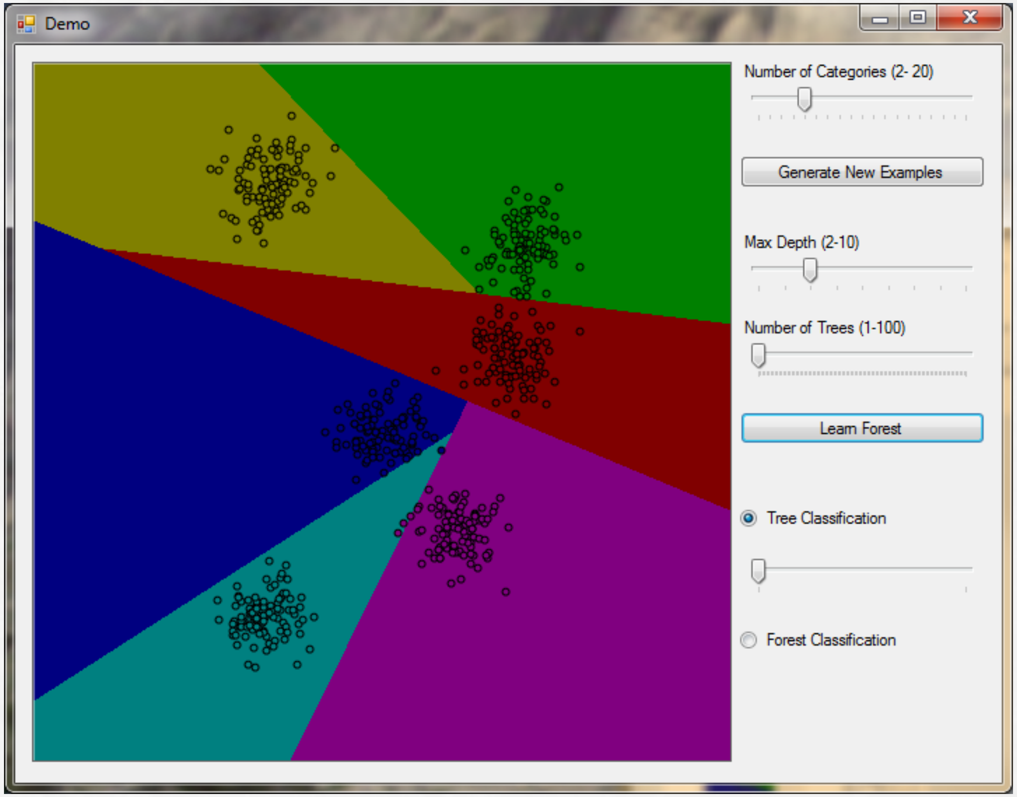
With a depth 2 tree, you cannot separate all six classes.

Toy Forest Classification Demo



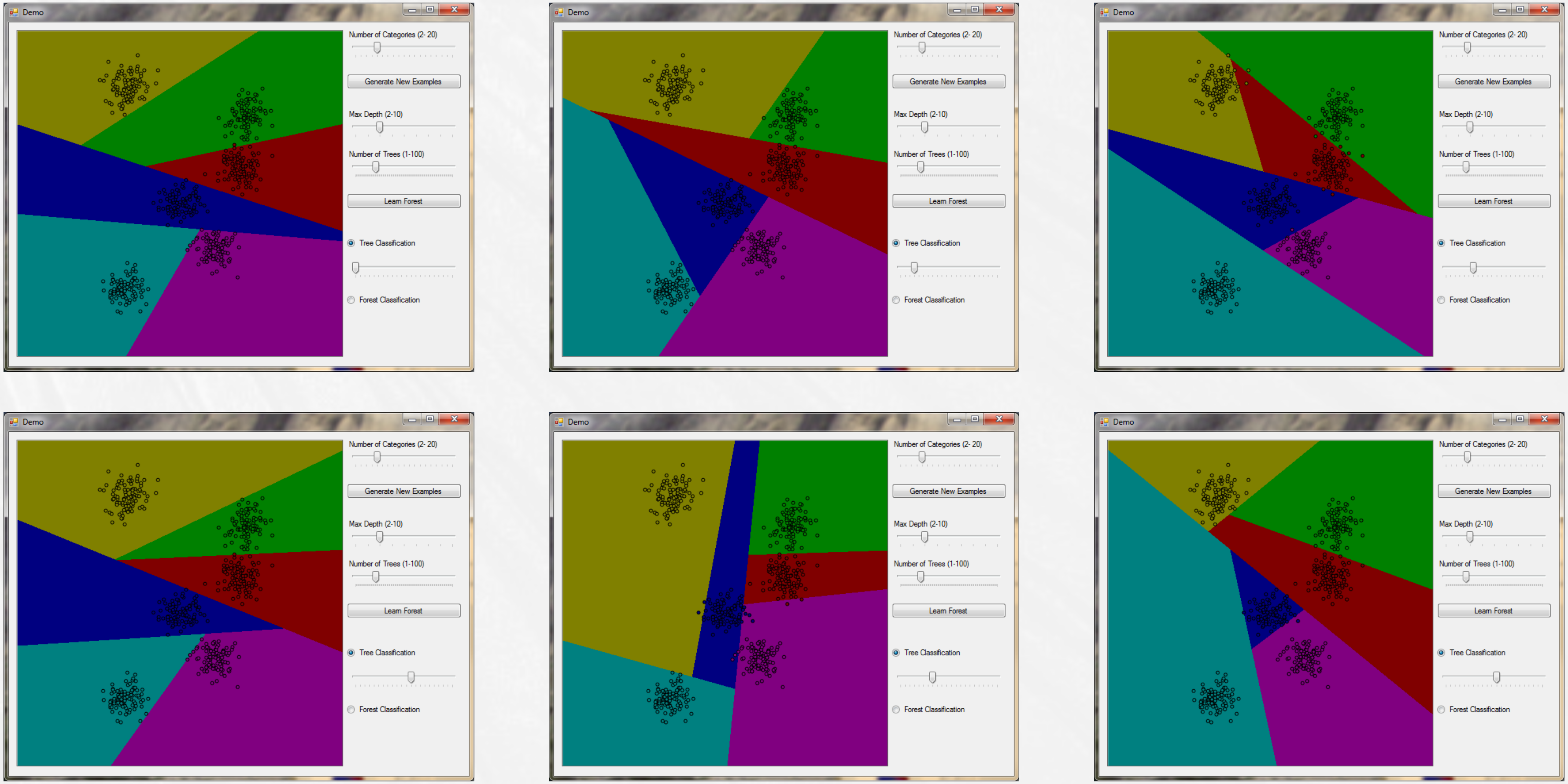
With a depth 3 tree, you are doing better, but still cannot separate all six classes.

Toy Forest Classification Demo



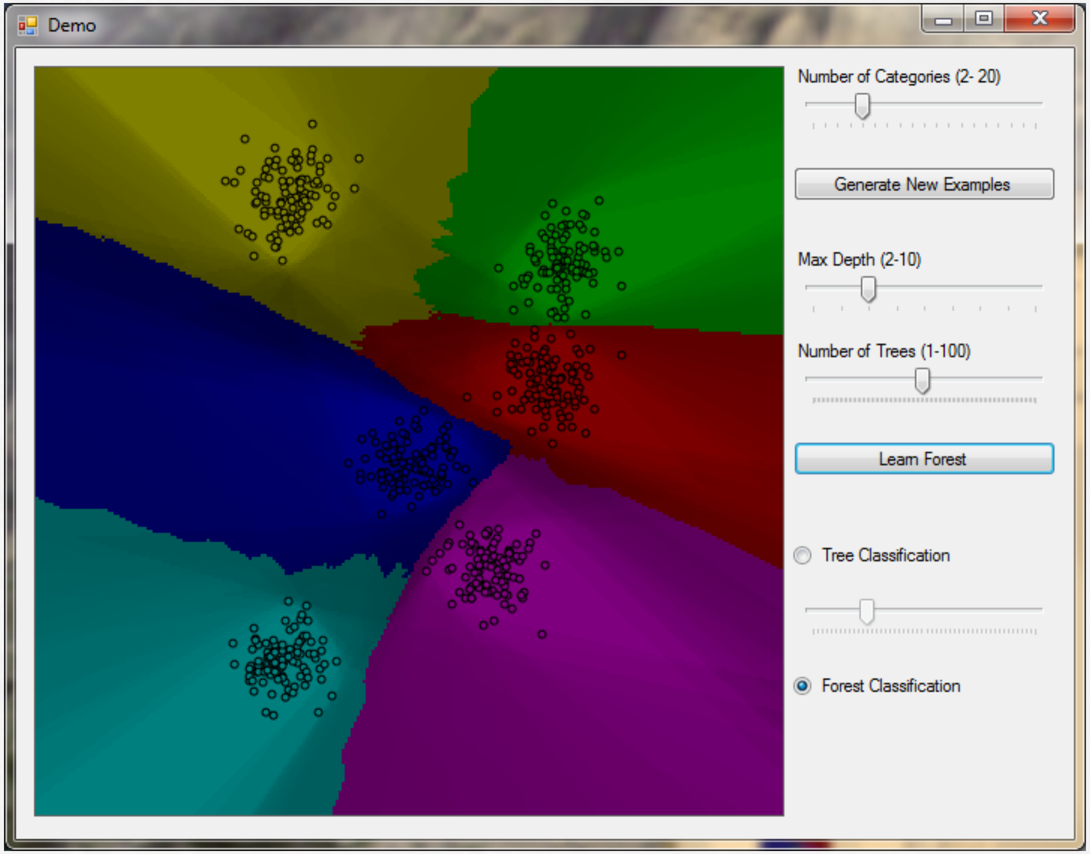
With a depth 4 tree, you now have at least as many leaf nodes as classes, and so are able to classify most examples correctly.

Toy Forest Classification Demo



Different trees within a forest can give rise to very different decision boundaries, none of which is particularly good on its own.

Toy Forest Classification Demo



Number of Categories (2- 20)

Max Depth (2-10)

Number of Trees (1-100)

○ Tree Classification

● Forest Classification

But averaging together many trees in a forest can result in decision boundaries that look very sensible, and are even quite close to the max margin classifier. (Shading represents entropy – darker is higher entropy).