CS 2750: Machine Learning

Ensemble Methods
Decision Trees

Prof. Adriana Kovashka
University of Pittsburgh
March 16, 2017

Plan for Today

- Ensemble methods
  - Bagging
  - Boosting
- Boosting application: Face detection
- Decision trees

Learning Ensembles

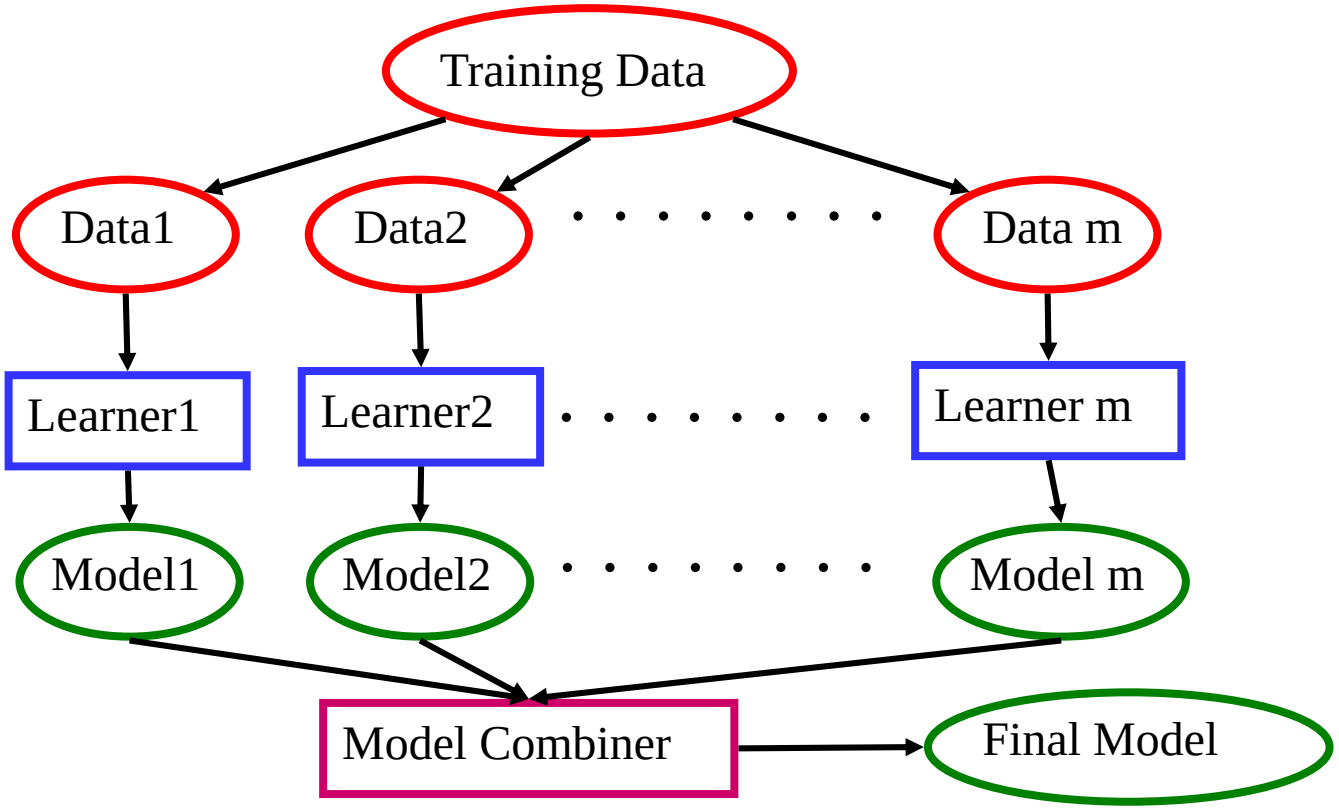
- Learn multiple alternative definitions of a concept using different training data or different learning algorithms
- Train several classifiers: SVM, KNN, logistic regression, decision tree, neural network etc.
- Call these classifiers $f_1(x)$, $f_2(x)$, …, $f_M(x)$
- Take majority of predictions:

  $$y = \text{majority}(f_1(x), f_2(x), \ldots, f_M(x))$$

- For regression use mean or median of the predictions
- Averaging is a form of regularization: each model can individually overfit but the average is able to overcome the overfitting

Learning Ensembles

- Learn multiple alternative definitions of a concept using different training data or different learning algorithms
- Combine decisions of multiple definitions



Value of Ensembles

- When combing multiple ***independent*** and ***diverse*** decisions each of which is at least more accurate than random guessing, random errors cancel each other out, correct decisions are reinforced.
- Human ensembles are demonstrably better
  - How many jelly beans in the jar?: Individual estimates vs. group average.
  - Who Wants to be a Millionaire: Expert friend vs. audience vote.
    - http://www.telegraph.co.uk/culture/books/3620109/Always-ask-the-audience.html

Adapted from Ray Mooney

Homogenous Ensembles

- Use a single, arbitrary learning algorithm but manipulate training data to make it learn multiple models.
  - Data1 ≠ Data2 ≠ … ≠ Data m
  - Learner1 = Learner2 = … = Learner m
- Different methods for changing training data:
  - Bagging: Resample training data
  - Boosting: Reweight training data

Bagging

- Create ensembles by repeatedly randomly resampling the training data (Brieman, 1996).
- Given a training set of size *n*, create *m* samples of size *n* by drawing *n* examples from the original data, ***with replacement***.
- Train *m* models and combine them using simple majority vote.
- Decreases error by decreasing the variance in the results due to ***unstable (high-variance) learners***, algorithms (like decision trees) whose output can change dramatically when the training data is slightly changed.
- However, often the errors of the different models are correlated, which defies the purpose of bagging.

Adapted from Ray Mooney

Boosting

- Originally proposed in (Schapire, 1990), revised to be a practical algorithm, AdaBoost, for building ensembles that empirically improves generalization performance (Freund & Shapire, 1996).
- Relies on ***weak learners*** which only need to generate a hypothesis with a training accuracy greater than 0.5.
- Examples are given weights. At each iteration, a new hypothesis is learned and the examples are reweighted to focus the system on examples that the most recently learned classifier got wrong.

Adapted from Ray Mooney

Boosting: Basic Algorithm

- General Loop:

  Set all examples to have equal uniform weights.

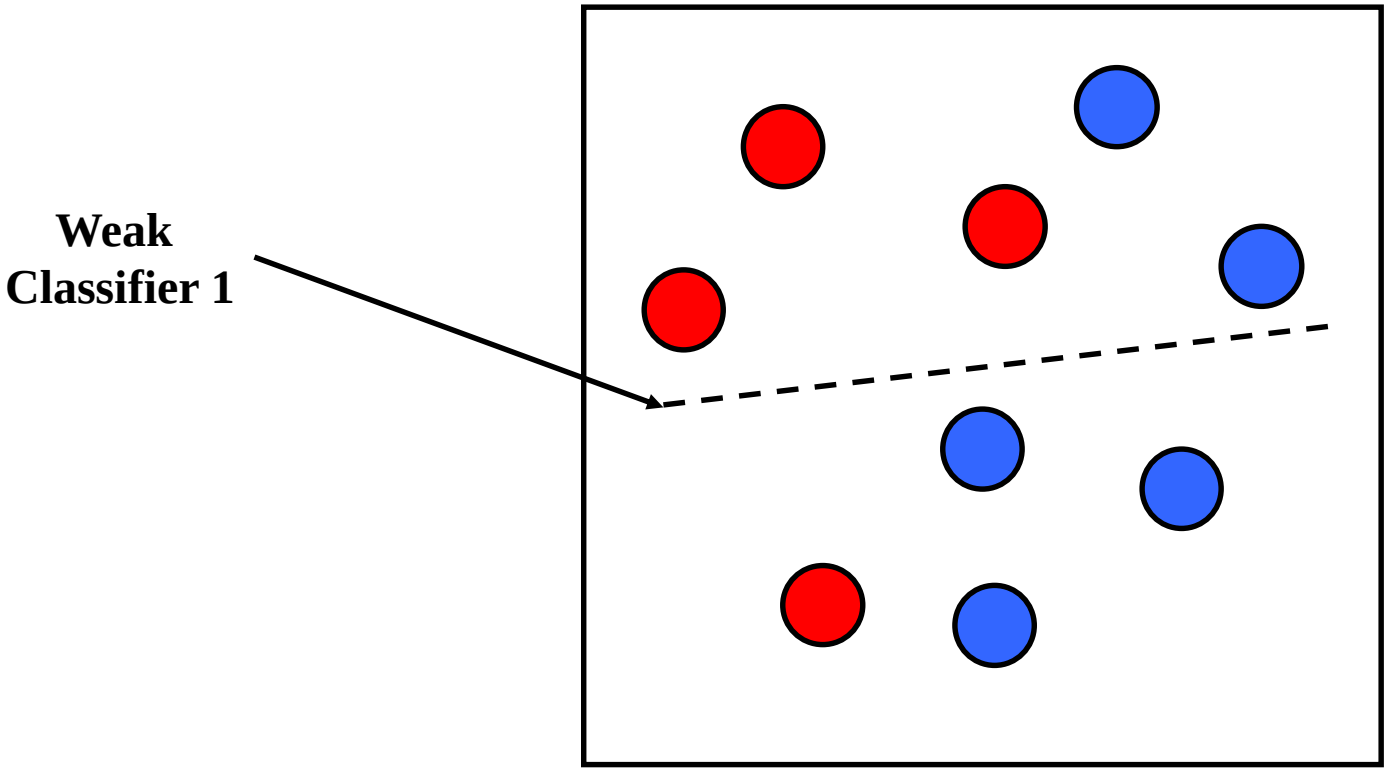
  For m from 1 to M do:

  Find the weak learner h_m that achieves lowest *weighted* training error

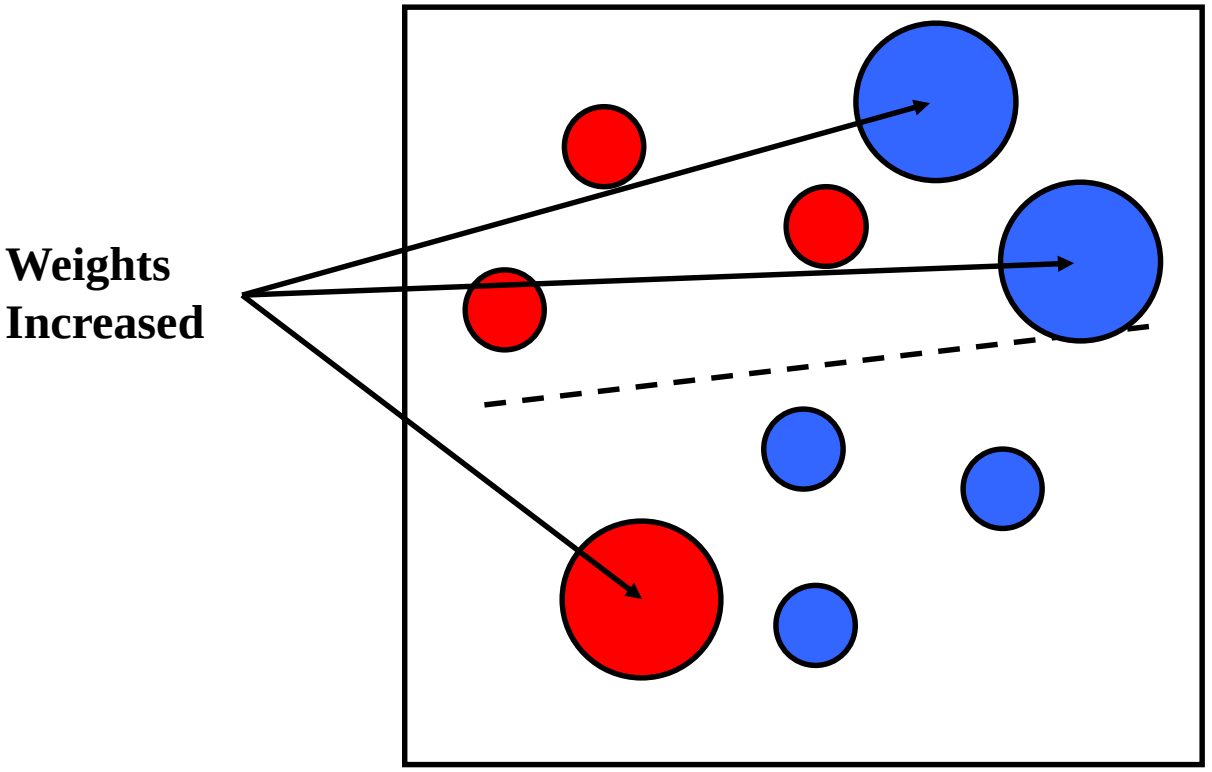
  Increase the weights of examples that h_m classifies incorrectly

- During testing: Each of the M classifiers gets a weighted vote proportional to its accuracy on the training data; final classifier is a linear combination of all weak learners.
- Base (weak) learner must focus on correctly classifying the most highly weighted examples while strongly avoiding over-fitting.
- Weak learners must perform better than chance.

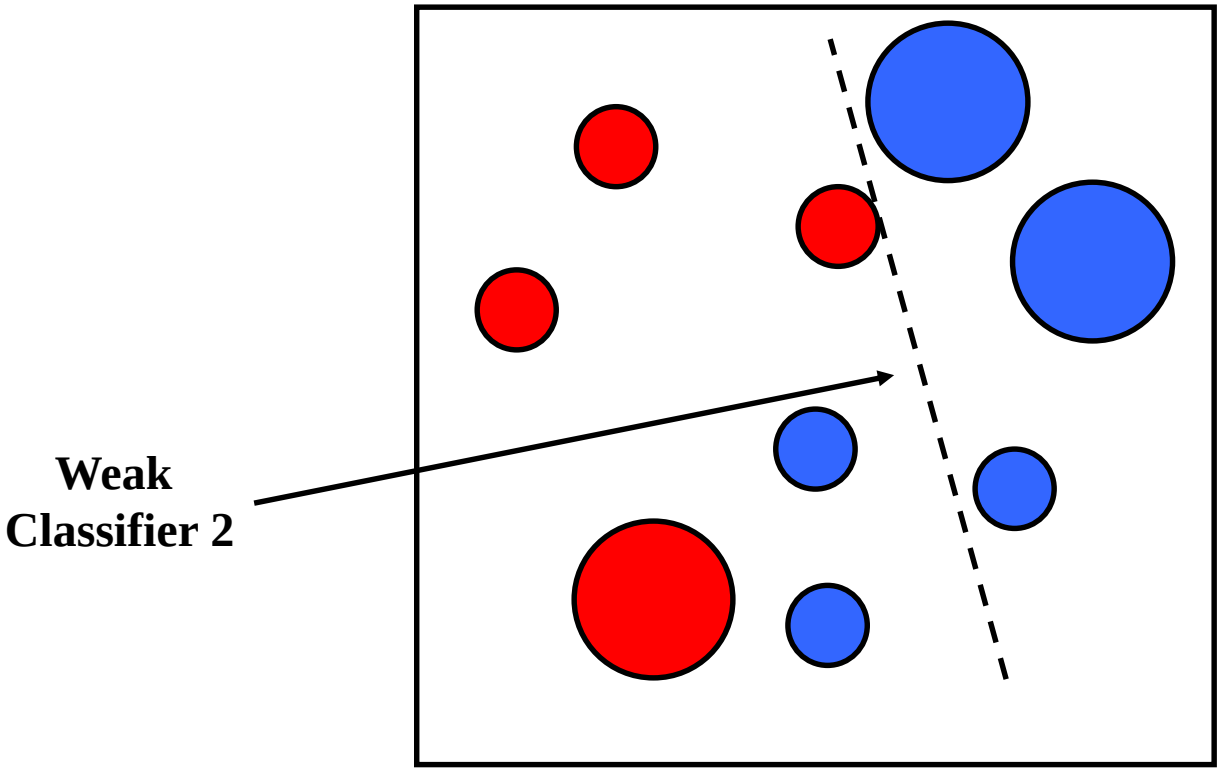
Boosting Illustration



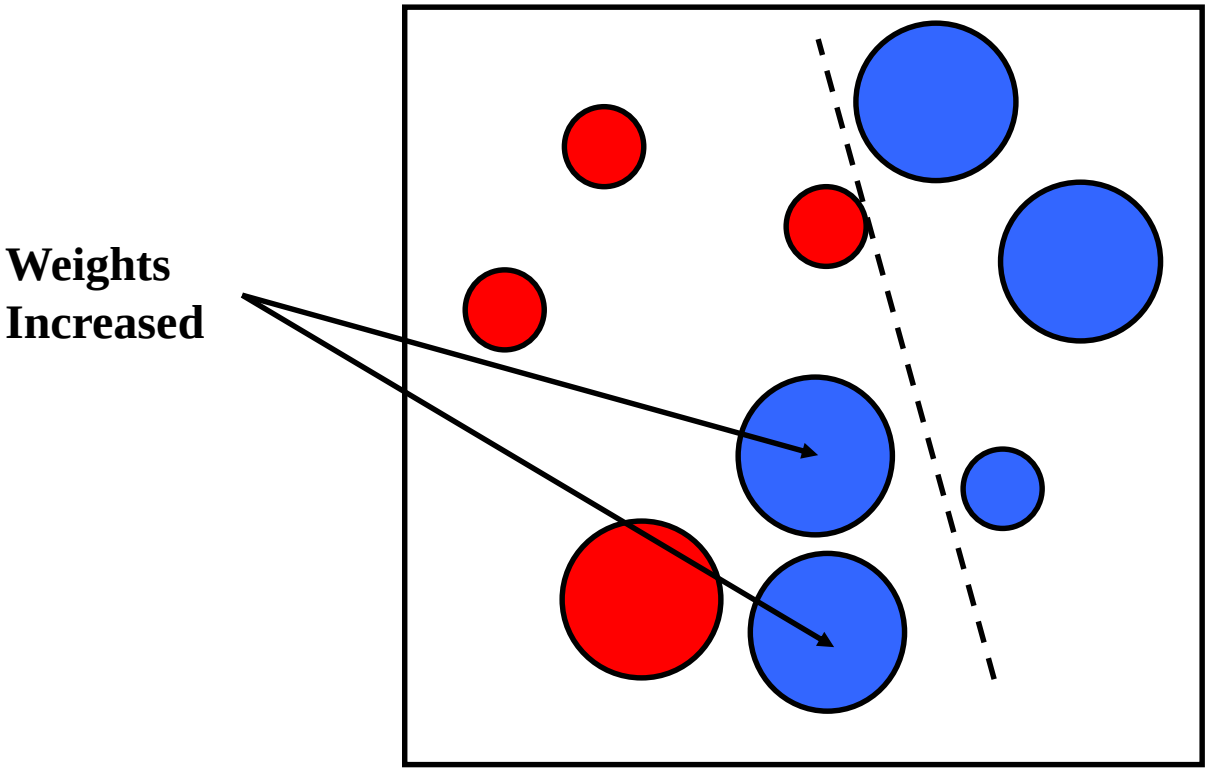
Boosting Illustration



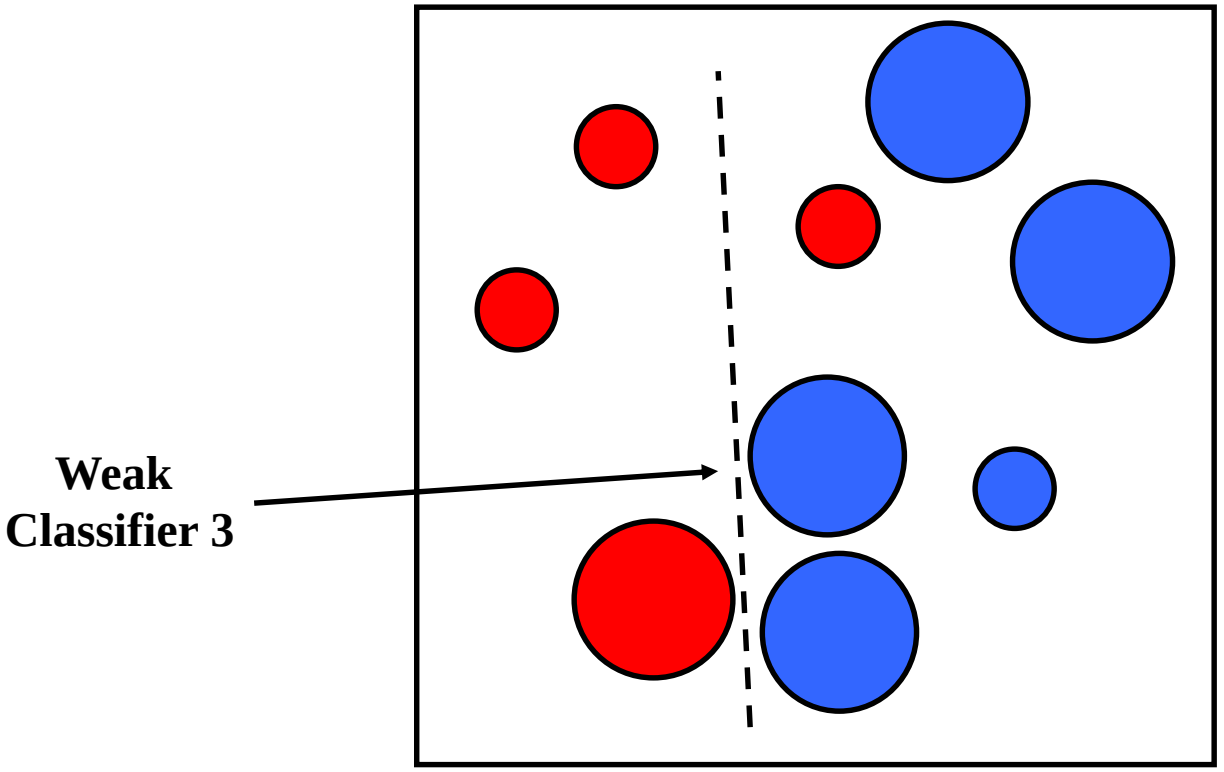
Boosting Illustration



Boosting Illustration

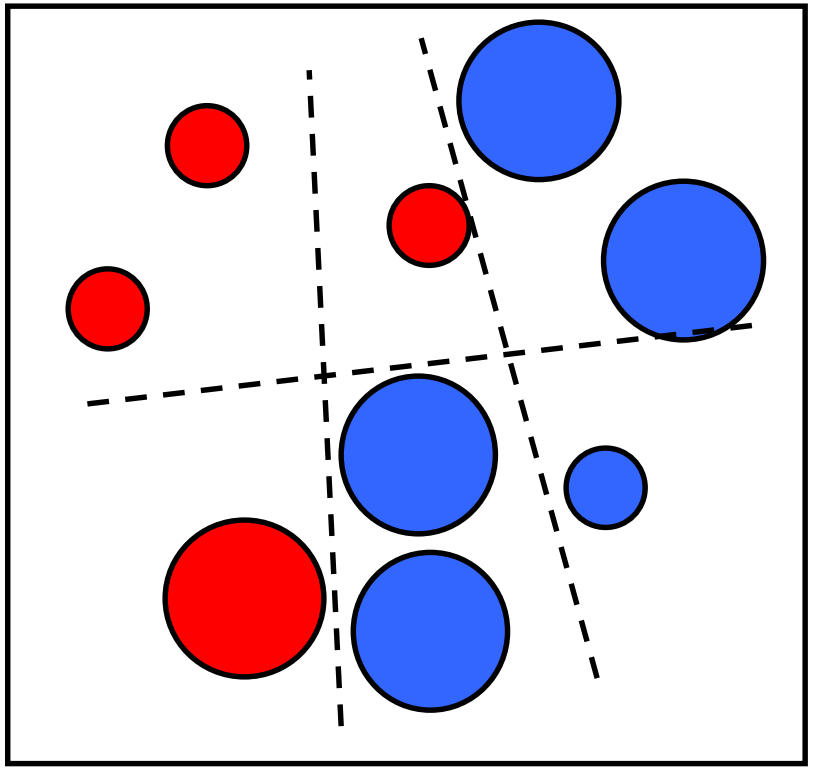


Boosting Illustration



Boosting Illustration

Final classifier is a combination of weak classifiers



AdaBoost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.15}$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}$$

   and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}. \tag{14.17}$$

   (c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp \{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \tag{14.18}$$

   (d) Normalize the weights so they sum to 1

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}) \right). \tag{14.19}$$

Start with uniform weights on training examples.

For *M* rounds

Evaluate *weighted* error for each weak learner, pick best learner.

$y_m(\boldsymbol{x}_n)$ is the prediction, t_n is ground truth for $\boldsymbol{x}_n$

Re-weight the examples: Incorrectly classified get more weight.

Final classifier is combination of weak ones, weighted according to error they had.

Figure from C. Bishop, notes from K. Grauman

Another Version of the AdaBoost Pseudocode

TrainAdaBoost(D, BaseLearn)
- For each example d_i in D let its weight w_i=1/|D|
- Let H be an empty set of hypotheses
- For t from 1 to T do:
  - Learn a hypothesis, h_t, from the weighted examples: h_t=BaseLearn(D)
  - Add h_t to H
  - Calculate the error, ε_t, of the hypothesis h_t as the total sum weight of the examples that it classifies incorrectly.
  - If $\varepsilon_t > 0.5$ then exit loop, else continue.
  - Let $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$
  - Multiply the weights of the examples that h_t classifies correctly by β_t
  - Rescale the weights of all of the examples so the total sum weight remains 1.
- Return H

TestAdaBoost(*ex*, H)
- Let each hypothesis, h_t, in H vote for *ex*'s classification with weight $\log(1/\beta_t)$
- Return the class with the highest weighted vote total.

Experimental Results on Ensembles (Freund & Schapire, 1996; Quinlan, 1996)

- Ensembles have been used to improve generalization accuracy on a wide variety of problems.
- On average, Boosting provides a larger increase in accuracy than Bagging.
- Boosting on rare occasions can degrade accuracy.
- Bagging more consistently provides a modest improvement.

Boosting application: Face detection

Challenges of face detection

- Sliding window detector must evaluate tens of thousands of location/scale combinations
- Faces are rare: 0–10 per image
- A megapixel image has ~10^6 pixels and a comparable number of candidate face locations
- For computational efficiency, we should try to spend as little time as possible on the non-face windows

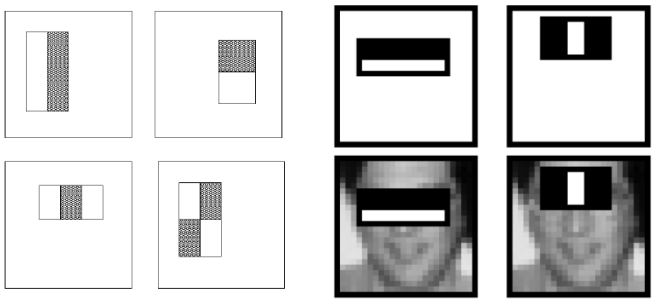
Viola-Jones face detector (CVPR 2001)

14,765 citations

Main idea:

- Represent local texture with efficiently computable "rectangular" features within window of interest
- Select discriminative features to be weak classifiers
- Use boosted combination of them as final classifier
- Form a cascade of such classifiers, rejecting clear negatives quickly (not discussed)

Viola-Jones detector: features

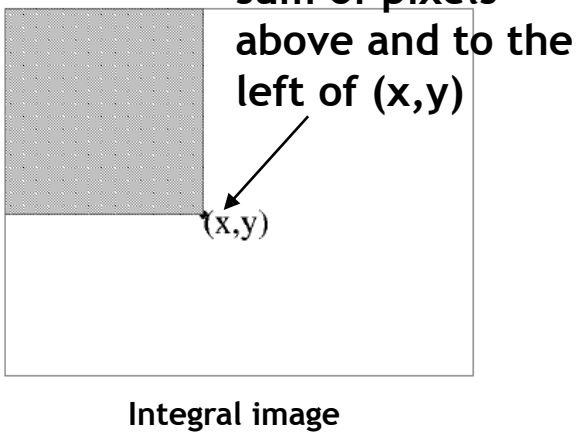


"Rectangular" filters

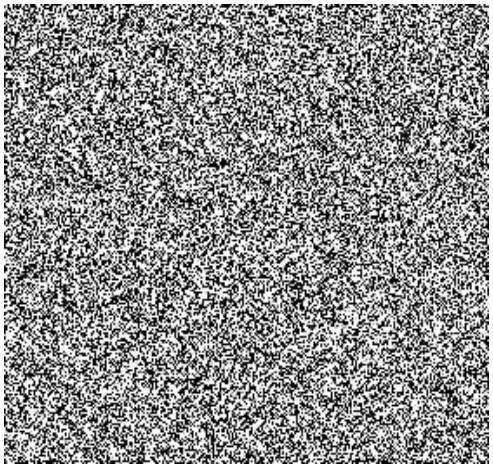
Feature output is difference between adjacent regions

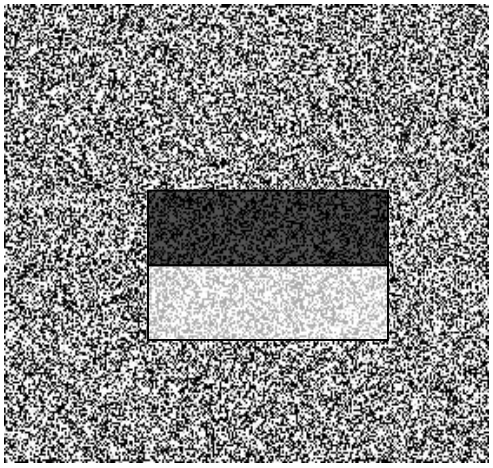
$$Value = \sum (pixels\ in\ white\ area) - \sum (pixels\ in\ black\ area)$$

Efficiently computable with integral image: any sum can be computed in constant time

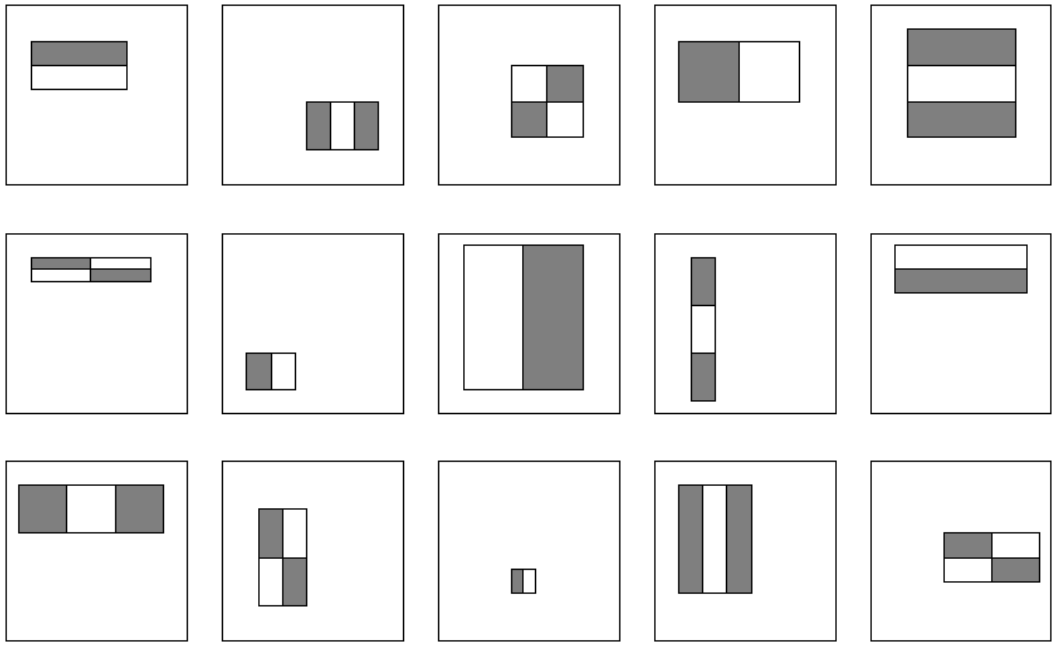


Example





Viola-Jones detector: features



Considering all possible filter parameters: position, scale, and type:

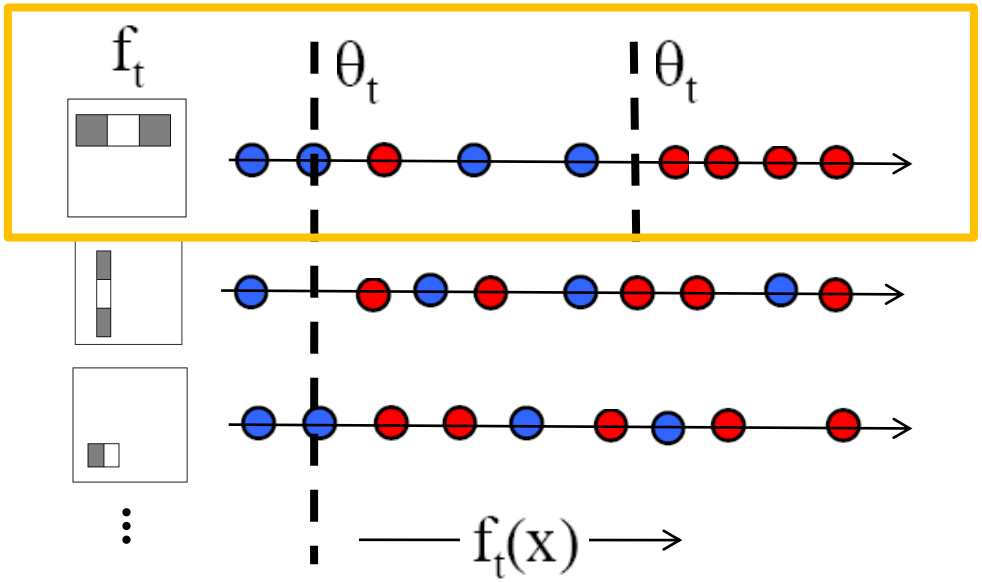
180,000+ possible features associated with each 24 x 24 window

Which subset of these features should we use to determine if a window has a face?

Use AdaBoost both to select the informative features and to form the classifier

Viola-Jones detector: AdaBoost

- **Want to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of *weighted* error.**



Outputs of a possible rectangle feature on faces and non-faces.

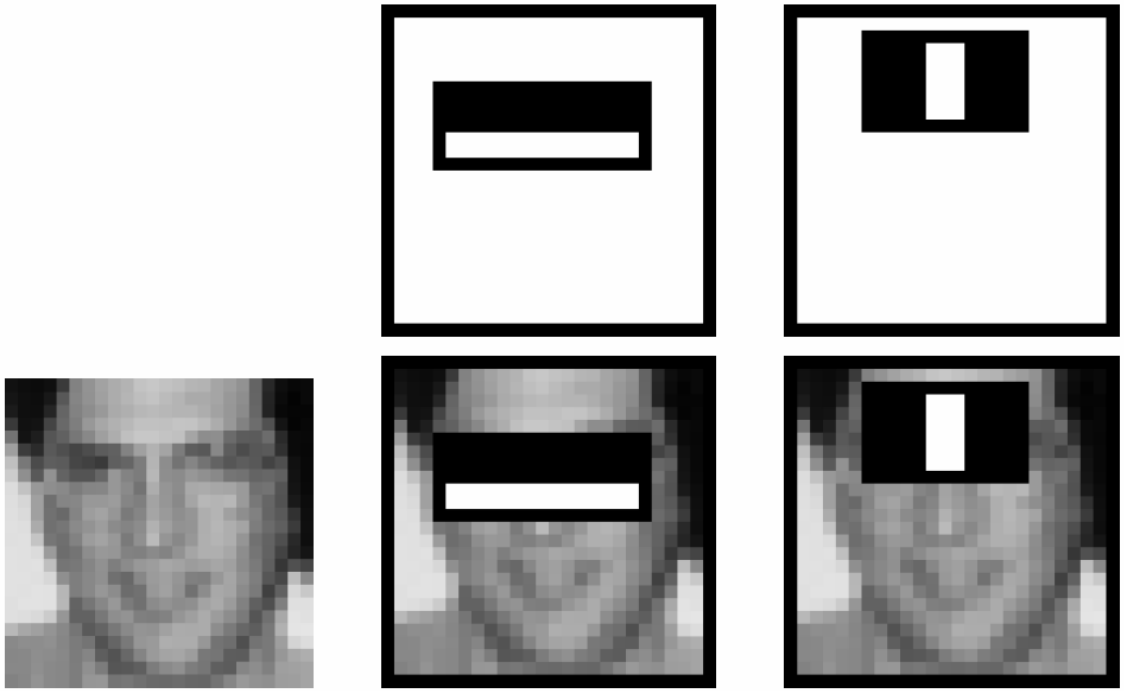
Resulting weak classifier:

$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$

For next round, reweight the examples according to errors, choose another filter/threshold combo.

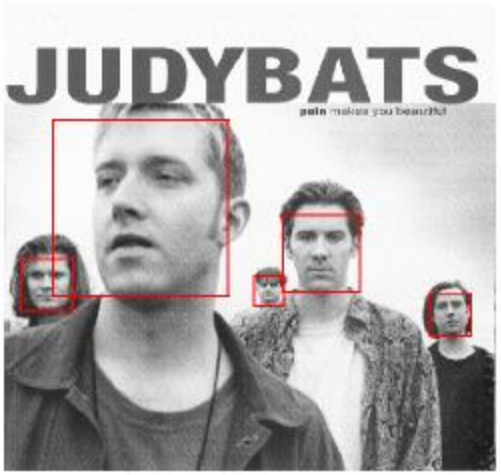
Boosting for face detection

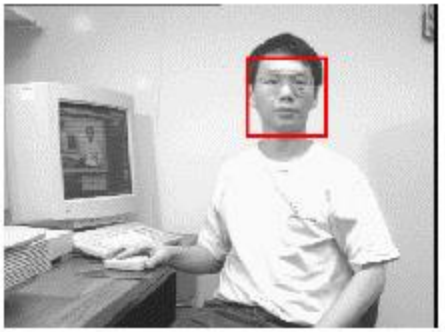
- First two features selected by boosting:

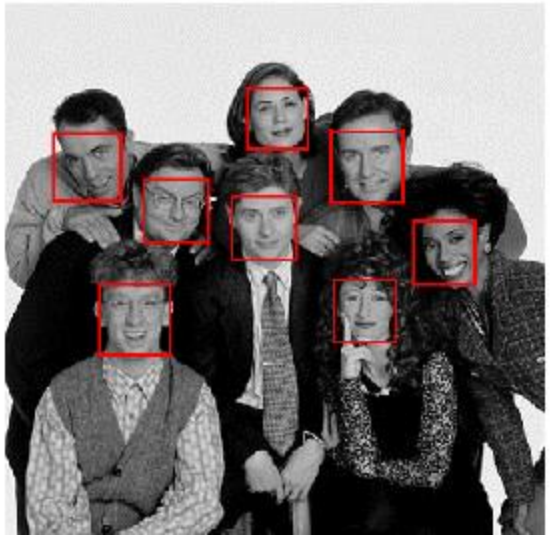


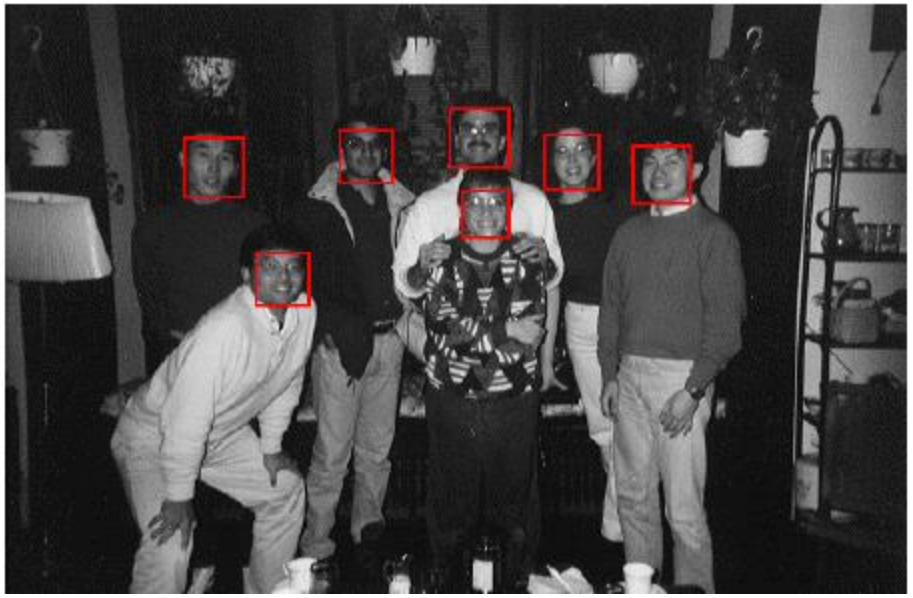
- This feature combination can yield 100% detection rate and 50% false positive rate

Viola-Jones face detector: Results

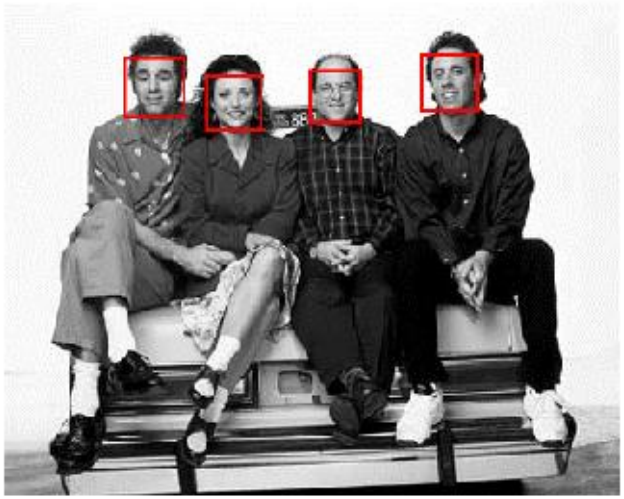


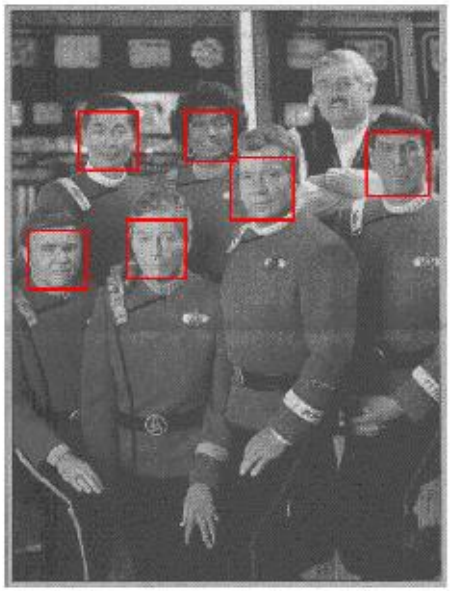


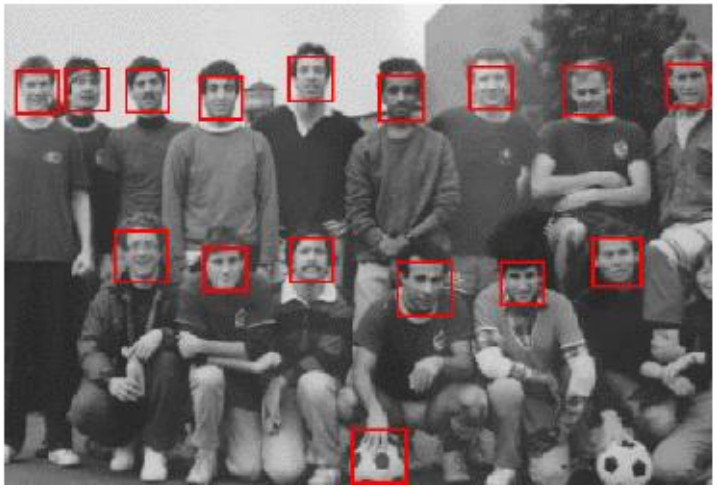


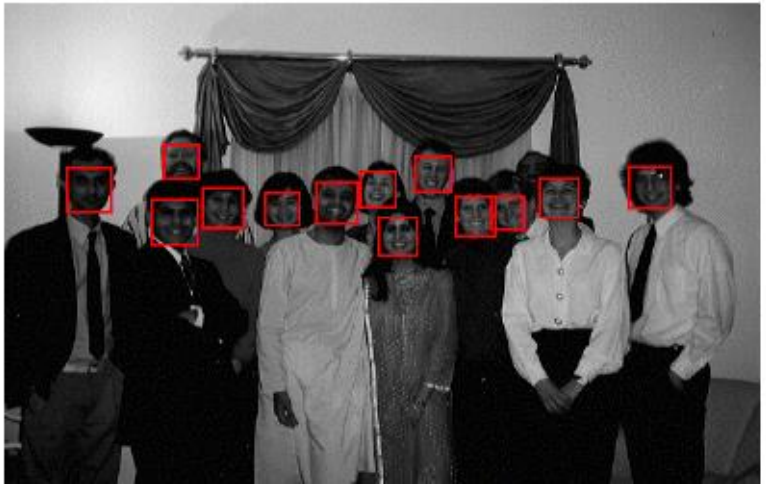


Viola-Jones face detector: Results









Decision Stumps

- Like the thresholded features=classifiers in face detection
- A single-level decision tree (discussed next)

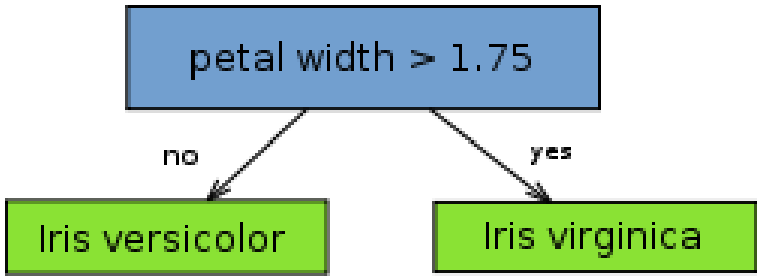


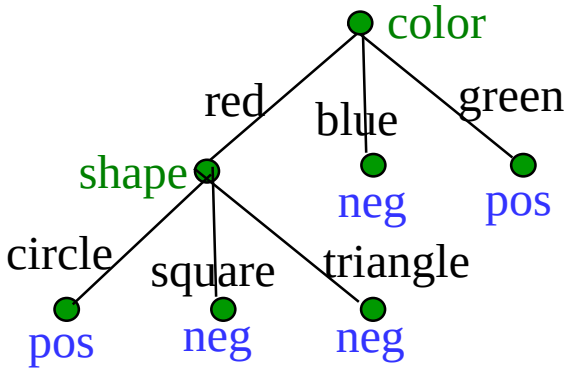
Figure from Wikipedia

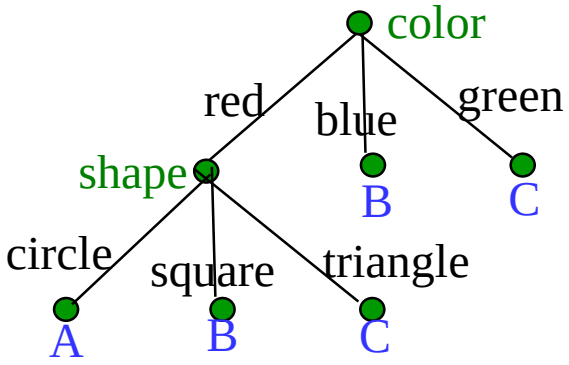
Plan for Today

- Ensemble methods
  - Bagging
  - Boosting
- Boosting application: Face detection
- Decision trees

Decision Trees

- Tree-based classifiers for instances represented as feature-vectors. Nodes test features, there is one branch for each value of the feature, and leaves specify the category.





- Can be rewritten as a set of rules:
  - red ∧ circle → pos
  - red ∧ circle → A
  - blue → B
  - red ∧ square → B
  - green → C
  - red ∧ triangle → C

Adapted from Ray Mooney

What about continuous features?

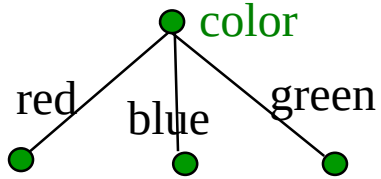
- Continuous (real-valued) features can be handled by allowing nodes to split a real valued feature into two ranges based on a threshold (e.g. length < 3 and length ≥ 3)
- Classification trees have discrete class labels at the leaves.

Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.

<big, red, circle>: +
<small, red, square>: –

<small, red, circle>: +
<big, blue, circle>: –



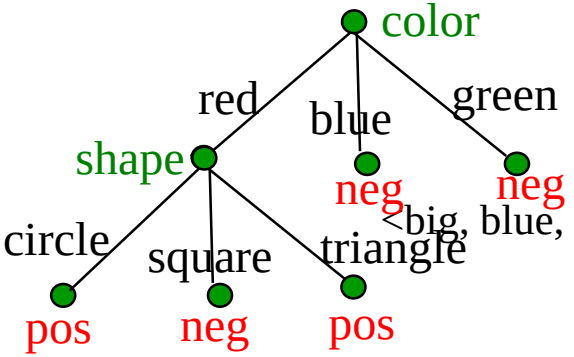
<big, red, circle>: +
<small, red, circle>: +
<small, red, square>: –

Top-Down Decision Tree Induction

- Recursively build a tree top-down by divide and conquer.

<big, red, circle>: +
<small, red, square>: –

<small, red, circle>: +
<big, blue, circle>: –



<big, red, circle>: +
<small, red, circle>: +
<small, red, square>: –

<big, blue, circle>: –

<big, red, circle>: +
<small, red, circle>: +

<small, red, square>: –

Decision Tree Induction Pseudocode

DTree(*examples*, *features*) returns a tree
  If all *examples* are in one category, return a leaf node with that category label.
  Else if the set of *features* is empty, return a leaf node with the category label that
      is the most common in examples.
  Else **pick a feature *F*** and create a node *R* for it
      For each possible value v_i of *F*:
          Add an out-going edge *E* to node *R* labeled with the value v_i.
          Let $examples_i$ be the subset of examples that have value v_i for *F*
          If $examples_i$ is empty
              then attach a leaf node to edge *E* labeled with the category that
                  is the most common in *examples*.
              else call DTree($examples_i$, *features* – {*F*}) and attach the resulting
                  tree as the subtree under edge *E*.
      Return the subtree rooted at *R*.

Adapted from Ray Mooney

Picking a Good Split Feature

- Goal is to have the resulting tree be as small as possible, per Occam's razor.
- Finding a minimal decision tree (nodes, leaves, or depth) is an NP-hard optimization problem.
- Top-down divide-and-conquer method does a greedy search for a simple tree but does not guarantee to find the smallest.
- Want to pick a feature that creates subsets of examples that are relatively "pure" in a single class so they are "closer" to being leaf nodes.
- There are a variety of heuristics for picking a good test, a popular one is based on information gain that originated with the ID3 system of Quinlan (1979).

Entropy

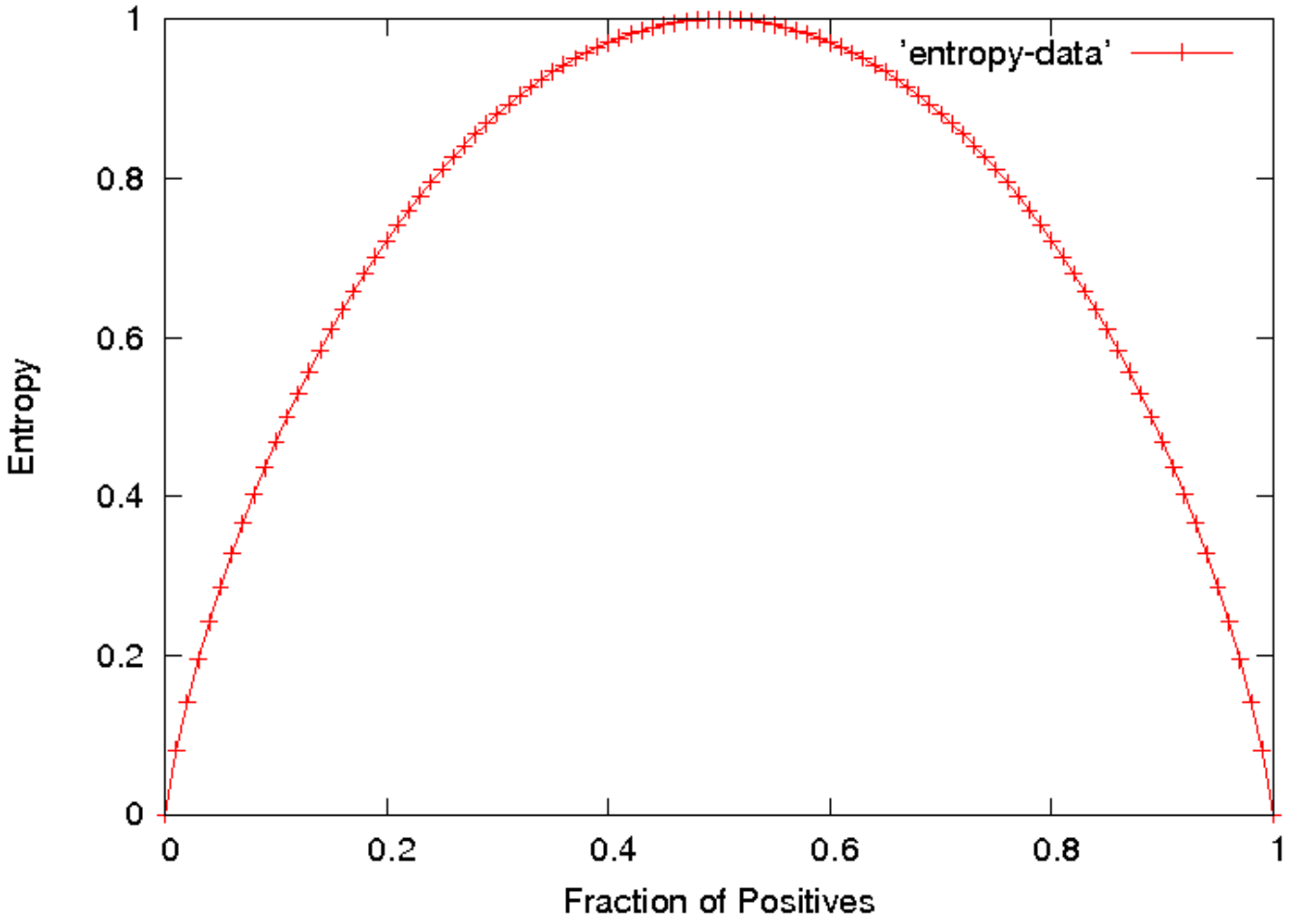
- Entropy (disorder, impurity) of a set of examples, S, relative to a binary classification is:

$$Entropy(S) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

  where p_1 is the fraction of positive examples in S and p_0 is the fraction of negatives.
- If all examples are in one category, entropy is zero (we define $0 \cdot \log(0)=0$)
- If examples are equally mixed ($p_1=p_0=0.5$), entropy is a maximum of 1.
- For multi-class problems with c categories, entropy generalizes to:

$$Entropy(S) = \sum_{i=1}^{c} - p_i \log_2(p_i)$$

Entropy Plot for Binary Classification



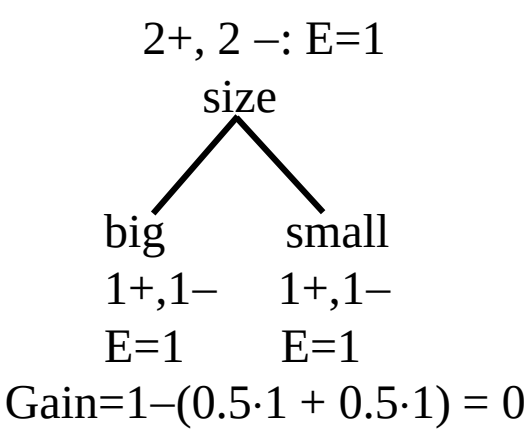
Information Gain

- The information gain of a feature F is the expected reduction in entropy resulting from splitting on this feature.

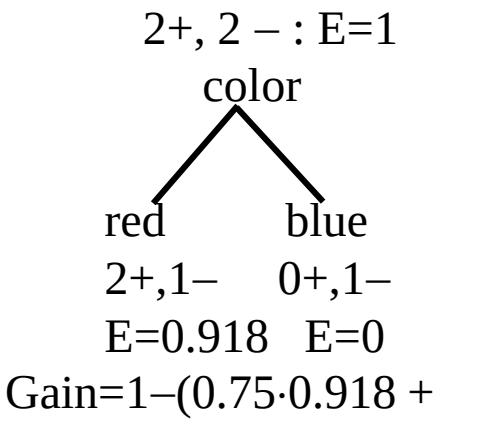
$$Gain(S,F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where S_v is the subset of S having value v for feature F.

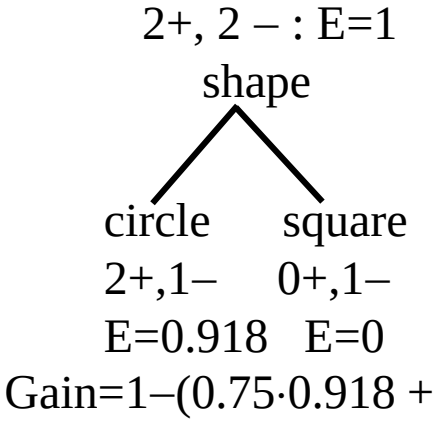
- Entropy of each resulting subset weighted by its relative size.
- Example:
  - <big, red, circle>: + <small, red, circle>: +
  - <small, red, square>: – <big, blue, circle>: –



Gain=1–(0.5·1 + 0.5·1) = 0



Gain=1–(0.75·0.918 + 0.25·0) = 0.311



Gain=1–(0.75·0.918 + 0.25·0) = 0.311

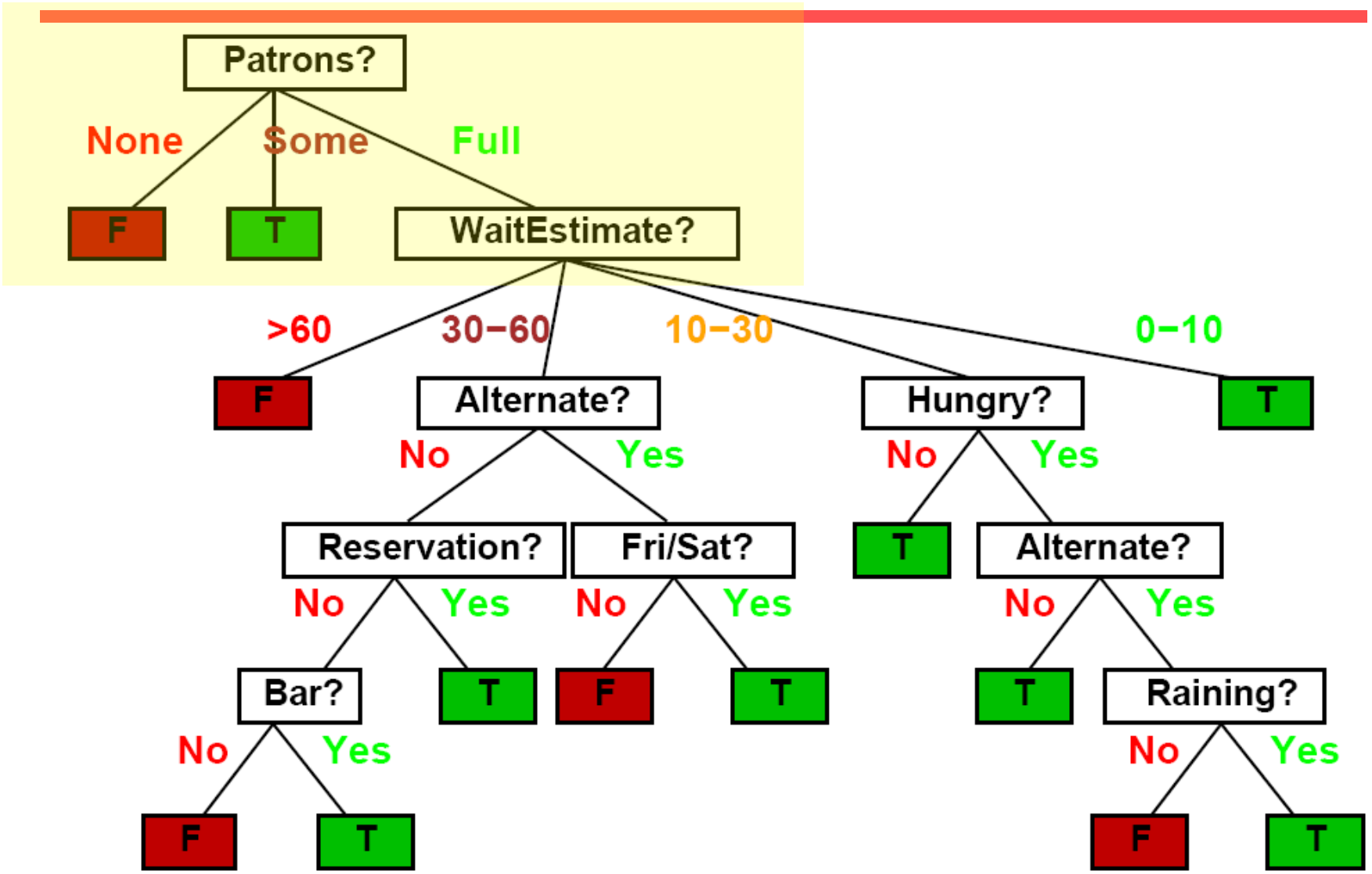
Another Example Decision Tree Classifier

- **Problem:** decide whether to wait for a table at a restaurant, based on the following attributes:
  1. **Alternate:** is there an alternative restaurant nearby?
  2. **Bar:** is there a comfortable bar area to wait in?
  3. **Fri/Sat:** is today Friday or Saturday?
  4. **Hungry:** are we hungry?
  5. **Patrons:** number of people in the restaurant (None, Some, Full)
  6. **Price:** price range ($, $$, $$$)
  7. **Raining:** is it raining outside?
  8. **Reservation:** have we made a reservation?
  9. **Type:** kind of restaurant (French, Italian, Thai, Burger)
  10. **WaitEstimate:** estimated waiting time (0-10, 10-30, 30-60, >60)

Another Example Decision Tree Classifier

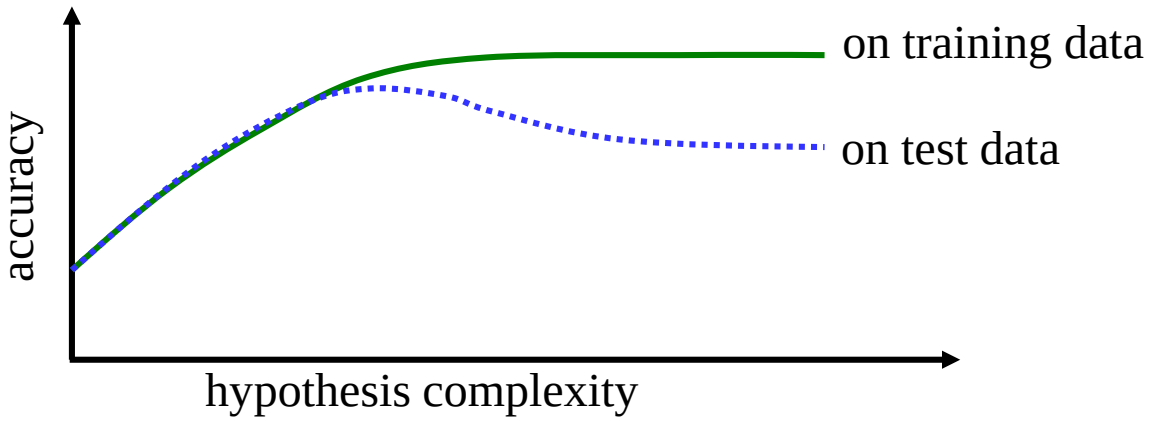
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Another Example Decision Tree Classifier



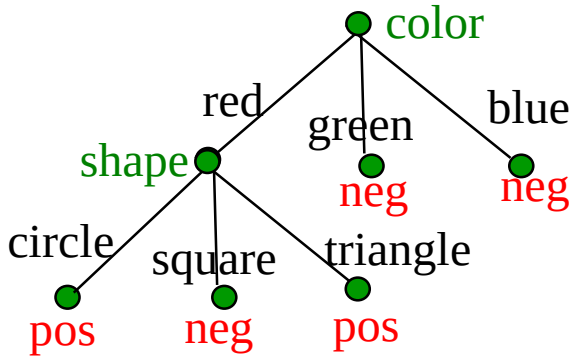
Overfitting

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  - There may be noise in the training data that the tree is erroneously fitting.
  - The algorithm may be making poor decisions towards the leaves of the tree that are based on very little data and may not reflect reliable trends.



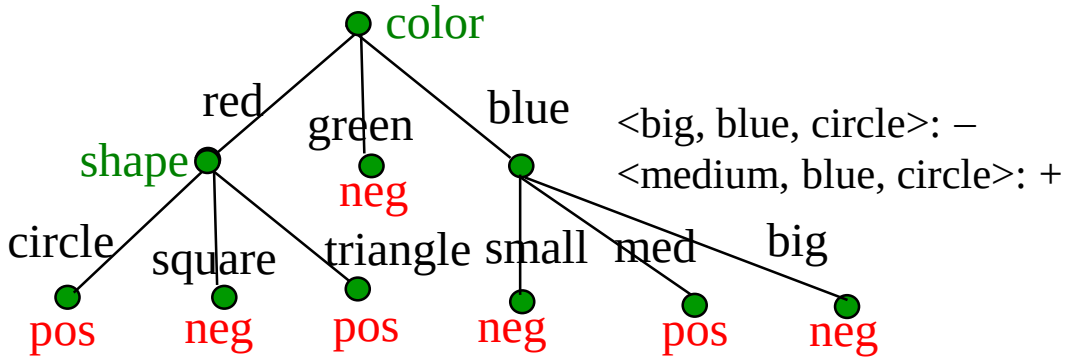
Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)



Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)



- Noise can also cause different instances of the same feature vector to have different classes. Impossible to fit this data and must label leaf with the majority class.
  - <big, red, circle>: neg (but really pos)

Overfitting Prevention (Pruning) Methods

- Two basic approaches for decision trees
  - Prepruning: Stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
  - Postpruning: Grow the full tree, then remove subtrees that do not have sufficient evidence.
- Label leaf resulting from pruning with the majority class of the remaining data.
- Some methods for determining which subtrees to prune:
  - Cross-validation: Reserve some training data as a hold-out set (validation set) to evaluate utility of subtrees.
  - Minimum description length (MDL): Determine if the additional complexity of the hypothesis is less complex than just explicitly remembering any exceptions resulting from pruning.

Reduced Error Pruning

- A post-pruning, cross-validation approach.

Partition training data in "grow" and "validation" sets.
Build a complete tree from the "grow" data.
Until accuracy on validation set decreases do:
    For each non-leaf node, n, in the tree do:
        Temporarily prune the subtree below n and replace it with a
            leaf labeled with the current majority class at that node.
        Measure and record the accuracy of the pruned tree on the validation set.
    Permanently prune the node that results in the greatest increase in accuracy on
        the validation set.