# Ensembles of Classifiers

Mark Craven and David Page
Computer Sciences 760
Spring 2018

www.biostat.wisc.edu/~craven/cs760/

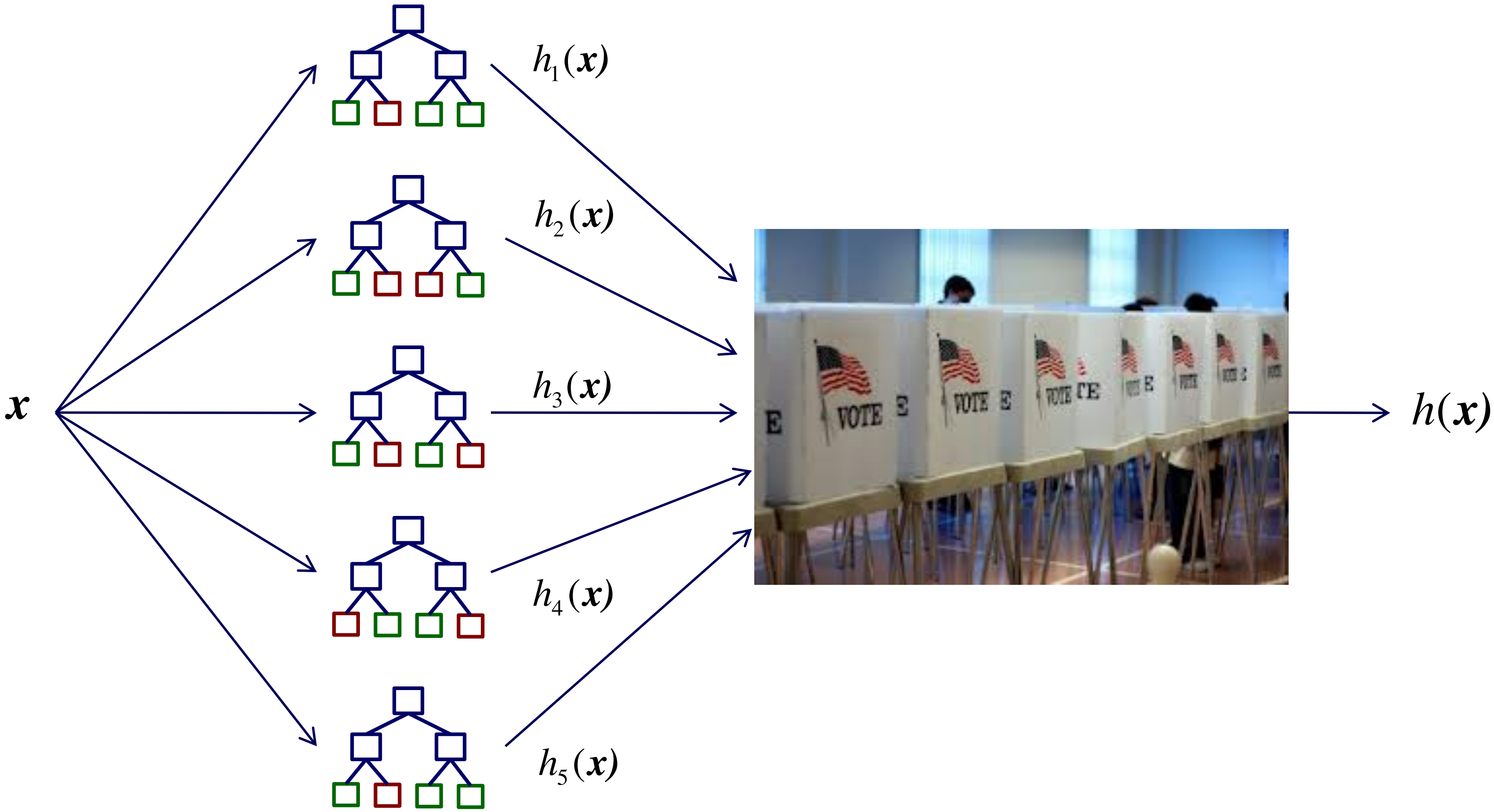
Some of the slides in these lectures have been adapted/borrowed from materials developed by Tom Dietterich, Pedro Domingos, Tom Mitchell, David Page, and Jude Shavlik

# Goals for the lecture

you should understand the following concepts

- ensemble
- bootstrap sample
- bagging
- boosting
- random forests
- error correcting output codes

# What is an ensemble?




a set of learned models whose individual decisions are combined in some way to make predictions for new instances

# When can an ensemble be more accurate?

- when the errors made by the individual predictors are (somewhat) uncorrelated, and the predictors' error rates are better than guessing (< 0.5 for 2-class problem)
- consider an idealized case…

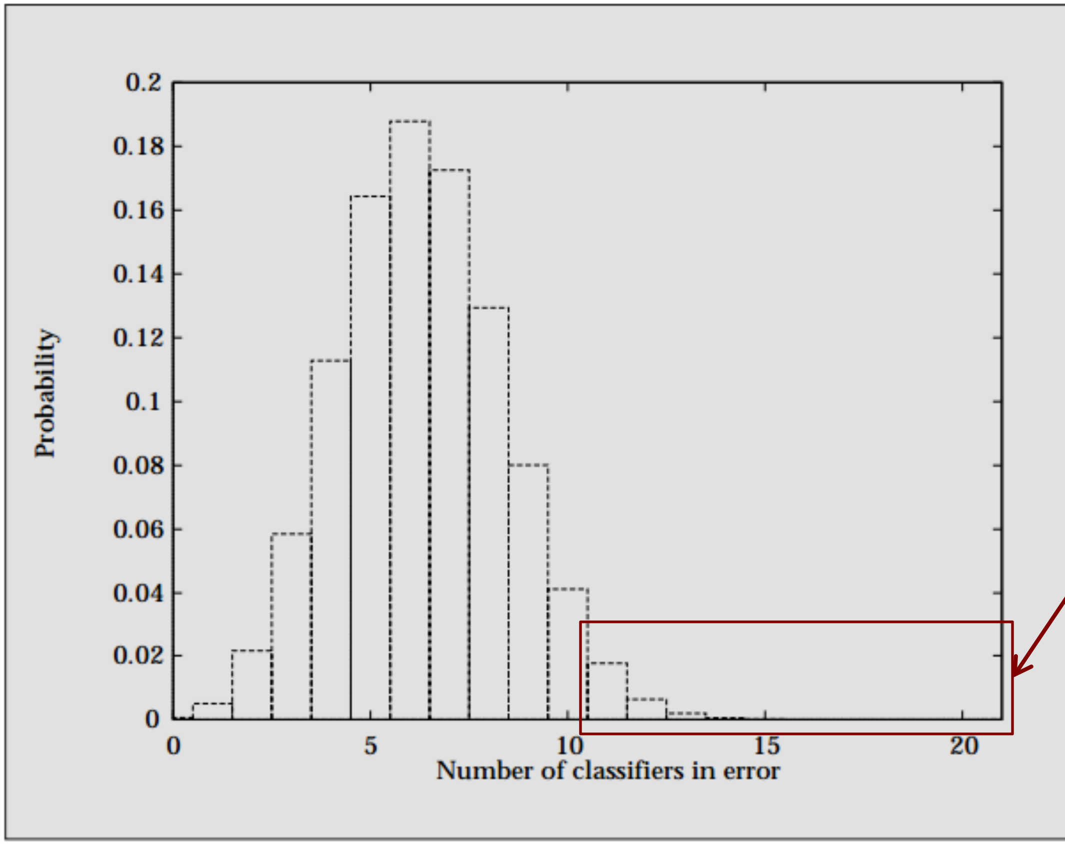



*Figure 1. The Probability That Exactly ℓ (of 21) Hypotheses Will Make an Error, Assuming Each Hypothesis Has an Error Rate of 0.3 and Makes Its Errors Independently of the Other Hypotheses.*

error rate of ensemble is represented by probability mass in this box = 0.026

Figure from Dietterich, *AI Magazine*, 1997

# How can we get diverse classifiers?

- In practice, we can’t get classifiers whose errors are completely uncorrelated, but we can encourage diversity in their errors by
  - choosing a variety of learning algorithms
  - choosing a variety of settings (e.g. # hidden units in neural nets) for the learning algorithm
  - choosing different subsamples of the training set (*bagging*)
  - using different probability distributions over the training instances (*boosting*)
  - choosing different features and subsamples (*random forests*)

# Bagging (Bootstrap Aggregation)

[Breiman, *Machine Learning* 1996]

learning:

given: learner $L$, training set $D = \{ \langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle \ \ldots \ \langle \boldsymbol{x}^{(m)}, y^{(m)} \rangle \}$

for $i \leftarrow 1$ to $T$ do

$D_\mathrm{i} \leftarrow m$ instances randomly drawn <u>with replacement</u> from $D$

$h_i \leftarrow$ model learned using $L$ on $D_\mathrm{i}$

classification:

given: test instance $\boldsymbol{x}$

predict $y \leftarrow$ plurality_vote( $h_1(\boldsymbol{x}) \ldots h_\mathrm{T}(\boldsymbol{x})$ )

regression:

given: test instance $\boldsymbol{x}$

predict $y \leftarrow$ mean( $h_1(\boldsymbol{x}) \ldots h_\mathrm{T}(\boldsymbol{x})$ )

# Bagging

- each sampled training set is a *bootstrap replicate*
  - contains $m$ instances (the same as the original training set)
  - on average it includes 63.2% of the original training set
  - some instances appear multiple times

- can be used with any base learner

- works best with *unstable* learning methods: those for which small changes in $D$ result in relatively large changes in learned models

# Empirical evaluation of bagging with C4.5

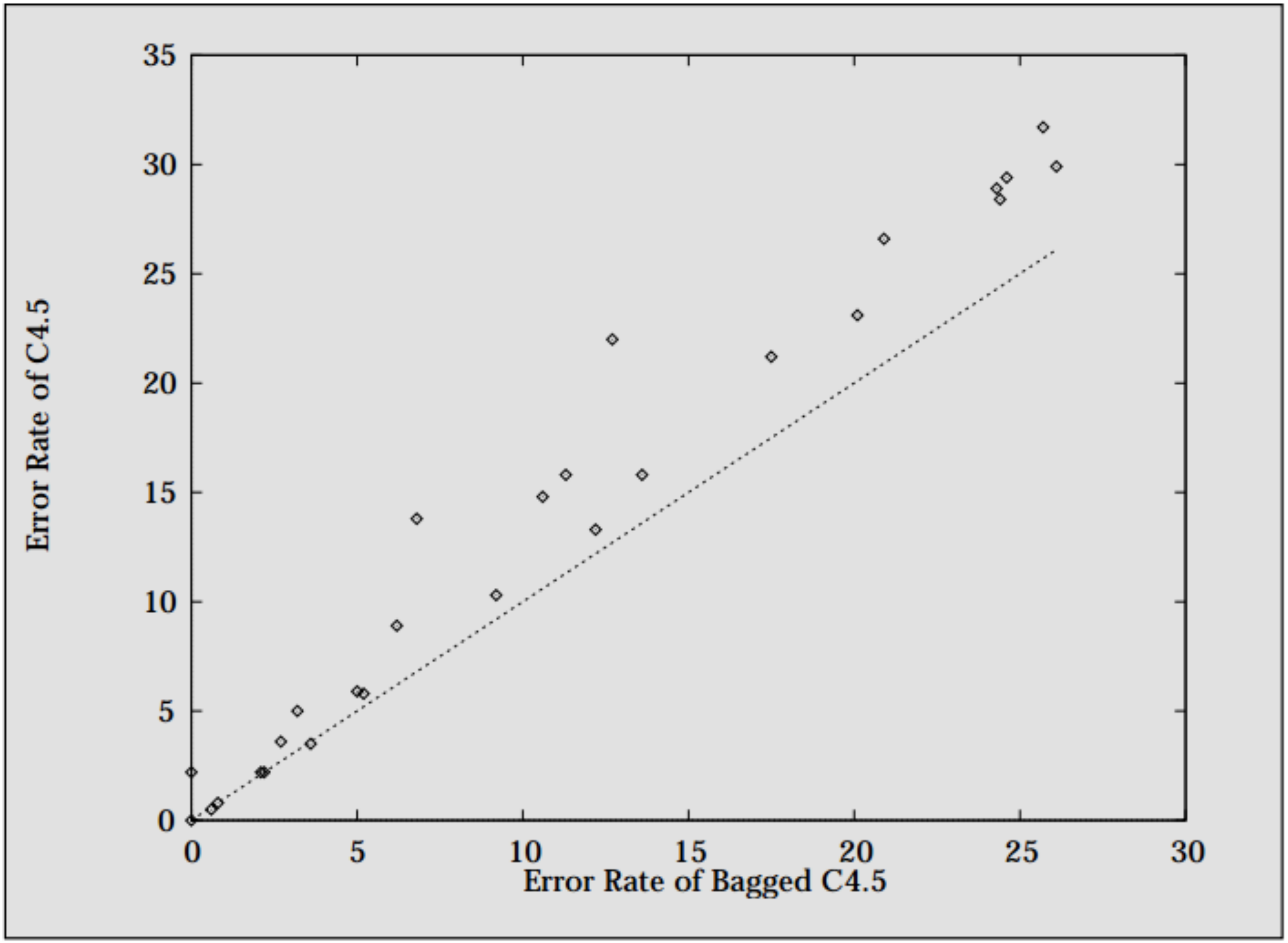



Figure from Dietterich, *AI Magazine*, 1997

Bagging reduced error of C4.5 on most data sets; wasn't harmful on any

# Boosting

- Boosting came out of PAC learning analysis
- A *weak PAC learning* algorithm is one that cannot PAC learn for arbitrary $\varepsilon$ and $\delta$, although its hypotheses are slightly better than random guessing
- Suppose we have a *weak PAC learning* algorithm $L$ for a concept class $C$. Can we use $L$ as a subroutine to create a strong PAC learner for $C$?
    - Yes, by boosting! [Schapire, *Machine Learning* 1990]
    - The original boosting algorithm was of theoretical interest, but assumed an unbounded source of training instances
- A later boosting algorithm, AdaBoost, has had notable practical success

# AdaBoost

[Freund & Schapire, Journal of Computer and System Sciences, 1997]

given: learner $L$, # stages $T$, training set $D = \{ \langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle \ \ldots \ \langle \boldsymbol{x}^{(m)}, y^{(m)} \rangle \}$

for all $i$ : $w_1(i) \leftarrow 1/m$ // initialize instance weights
for $t \leftarrow 1$ to $T$ do
  for all $i$ : $p_t(i) \leftarrow w_t(i) / (\sum_j w_t(j))$ // normalize weights
  $h_t \leftarrow$ model learned using $L$ on $D$ and $p_t$
  $\varepsilon_t \leftarrow \sum_i p_t(i)(1 - \delta(h_t(\boldsymbol{x}^{(i)}), y^{(i)}))$ // calculate weighted error
  if $\varepsilon_t > 0.5$ then
    $T \leftarrow t - 1$
    break
  $\beta_t \leftarrow \varepsilon_t / (1 - \varepsilon_t)$
  for all $i$ where $h_t(\boldsymbol{x}^{(i)}) = y^{(i)}$ // down-weight correct examples
    $w_{t+1}(i) \leftarrow w_t(i)\, \beta_t$

return:

$$h(\boldsymbol{x}) = \arg\max_y \sum_{t=1}^{T} \left( \log \frac{1}{\beta_t} \right) \delta\left(h_t(\boldsymbol{x}),\ y\right)$$

# Implementing weighted instances with AdaBoost

- AdaBoost calls the base learner $L$ with probability distribution $p_t$ specified by weights on the instances

- there are two ways to handle this
  1. Adapt $L$ to learn from weighted instances; straightforward for decision trees and naïve Bayes, among others
  2. Make a large ($\gg m$) unweighted set of instances by replicating each instance many times; sample this set according to $p_t$; run $L$ in the ordinary manner

# AdaBoost variants

- AdaBoost.M1: 1-of-n multiclass tasks
- AdaBoost.M2: arbitrary multiclass tasks
- AdaBoost.R: regression
- confidence-rated predictions (learners output their confidence in predicted class for each instance)
- etc.

# Empirical evaluation of boosting with C4.5

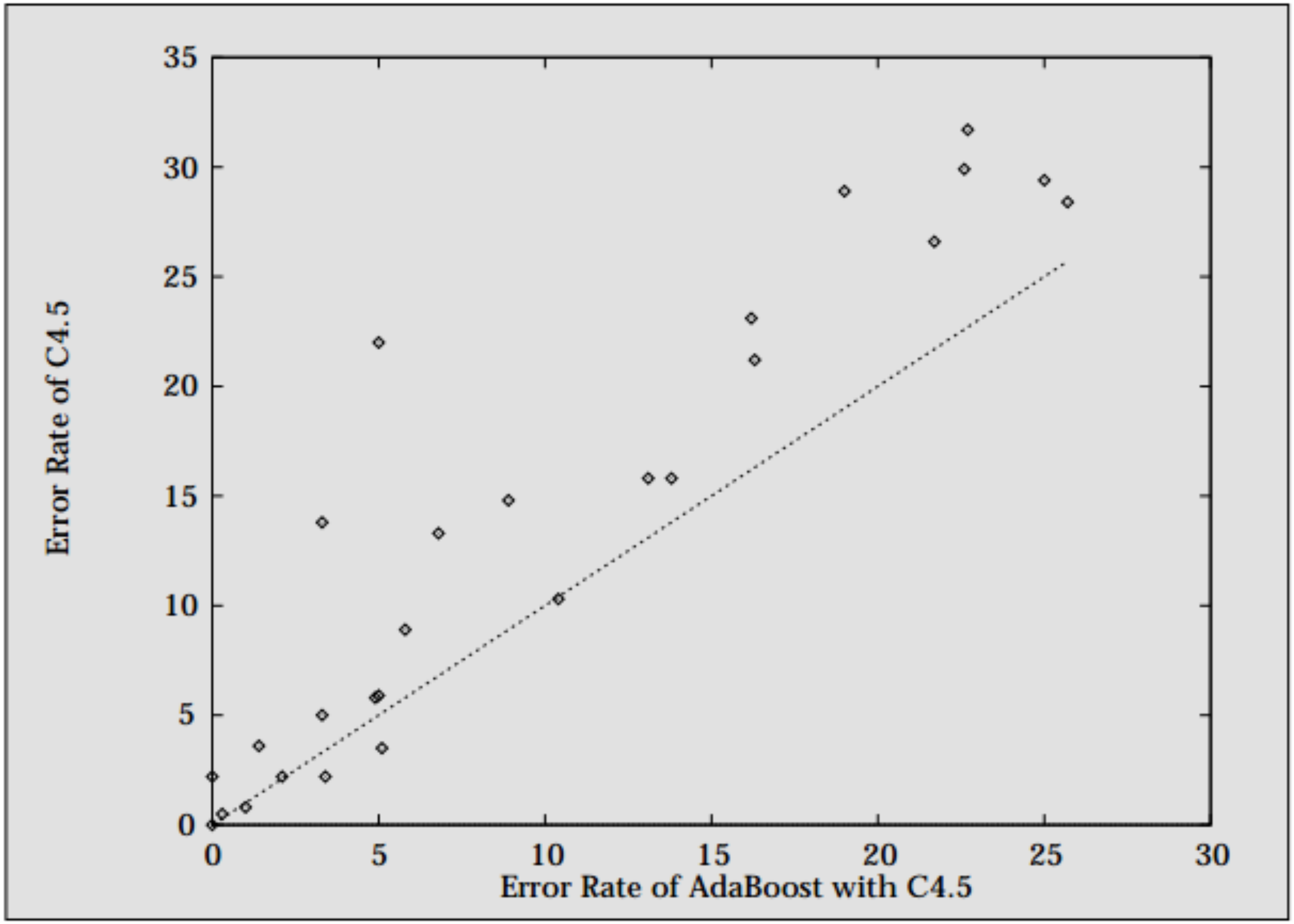



Figure from Dietterich, *AI Magazine*, 1997

# Bagging and boosting with C4.5

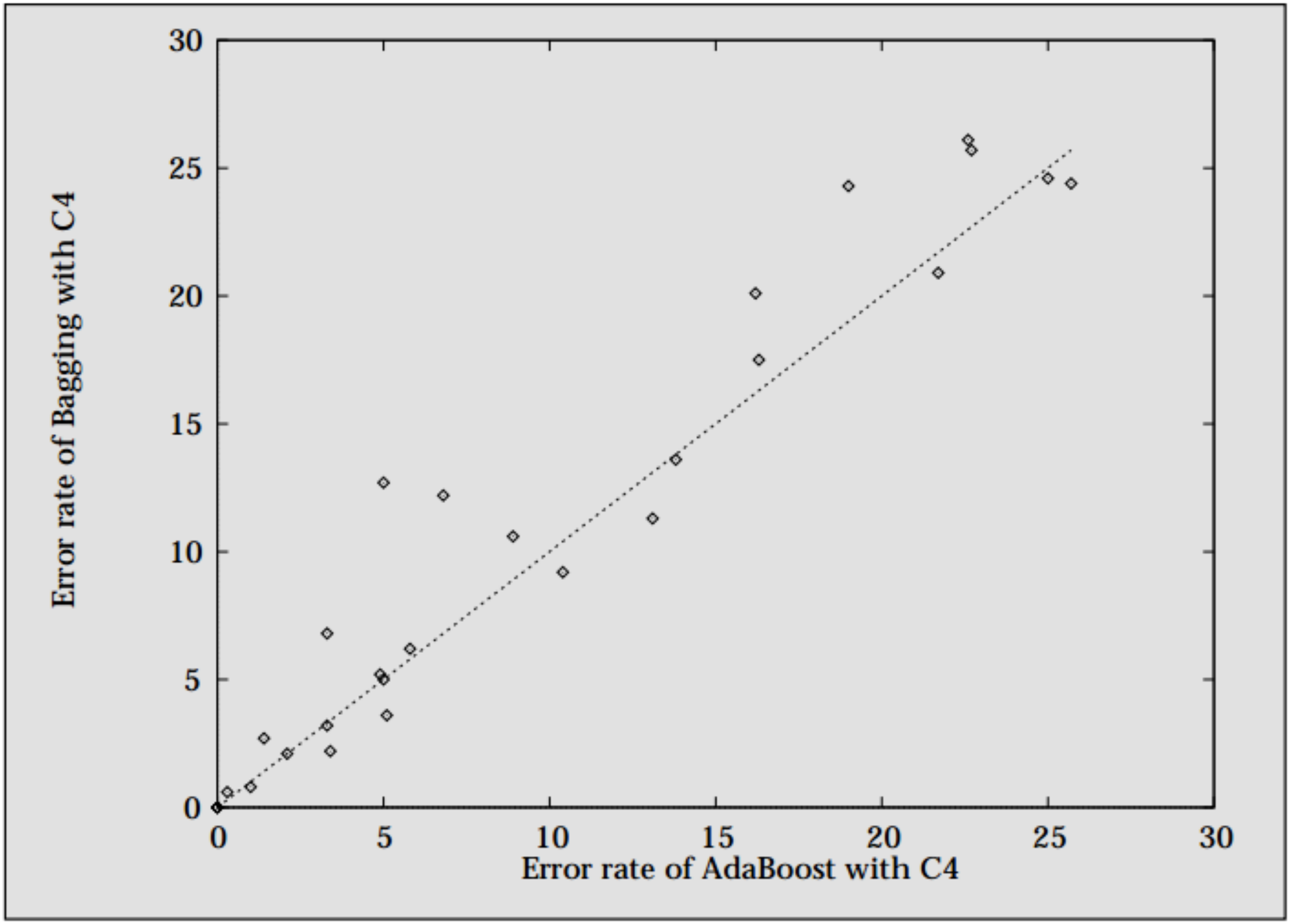



Figure from Dietterich, *AI Magazine*, 1997

# Empirical study of bagging vs. boosting

[Opitz & Maclin, *JAIR* 1999]

- 23 data sets
- C4.5 and neural nets as base learners
- bagging almost always better than single decision tree or neural net
- boosting can be much better than bagging
- however, boosting can sometimes reduce accuracy (too much emphasis on outliers?)

# Random forests

[Breiman, Machine Learning 2001]

given: candidate feature splits $F$,
training set $D = \{ \langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle \ \ldots \ \langle \boldsymbol{x}^{(m)}, y^{(m)} \rangle \}$
for $i \leftarrow 1$ to $T$ do
$D_{\mathrm{i}} \leftarrow m$ instances randomly drawn <u>with replacement</u> from $D$
$h_i \leftarrow$ <u>randomized</u> decision tree learned with $F$, $D_i$

randomized decision tree learning:
to select a split at a node
$R \leftarrow$ randomly select (without replacement) $f$ feature splits from $F$
(where $f \ll |F|$ )
choose the best feature split in $R$
do not prune trees

classification/regression:
as in bagging

# One large-scale empirical study
[Fernández-Delgado *JMLR* 2014]

- compared 179 classifiers on 121 data sets
- random forest was the best family of classifiers (3 classifiers in the top 5)

| Rank | Acc. | $\kappa$ | Classifier |
|---|---|---|---|
| **32.9** | 82.0 | 63.5 | parRF_t (RF) |
| 33.1 | **82.3** | **63.6** | rf_t (RF) |
| 36.8 | 81.8 | 62.2 | svm_C (SVM) |
| 38.0 | 81.2 | 60.1 | svmPoly_t (SVM) |
| 39.4 | 81.9 | 62.5 | rforest_R (RF) |
| 39.6 | 82.0 | 62.0 | elm_kernel_m (NNET) |
| 40.3 | 81.4 | 61.1 | svmRadialCost_t (SVM) |
| 42.5 | 81.0 | 60.0 | svmRadial_t (SVM) |
| 42.9 | 80.6 | 61.0 | C5.0_t (BST) |
| 44.1 | 79.4 | 60.5 | avNNet_t (NNET) |
| 45.5 | 79.5 | 61.0 | nnet_t (NNET) |
| 47.0 | 78.7 | 59.4 | pcaNNet_t (NNET) |
| 47.1 | 80.8 | 53.0 | BG_LibSVM_w (BAG) |
| 47.3 | 80.3 | 62.0 | mlp_t (NNET) |
| 47.6 | 80.6 | 60.0 | RotationForest_w (RF) |
| 50.1 | 80.9 | 61.6 | RRF_t (RF) |
| 51.6 | 80.7 | 61.4 | RRFglobal_t (RF) |
| 52.5 | 80.6 | 58.0 | MAB_LibSVM_w (BST) |
| 52.6 | 79.9 | 56.9 | LibSVM_w (SVM) |
| 57.6 | 79.1 | 59.3 | adaboost_R (BST) |

# One application of random forests: human pose recognition in the Xbox Kinect

[Shotton et al., *CVPR* 2011]

Classification task

- Given: a depth image
- Do: classify each pixel into one of 31 body parts

# Bias/variance and ensembles

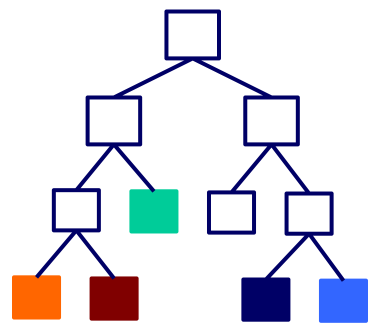
- bagging & random forests work mainly by reducing variance

- boosting works by
  - primarily reducing bias in the early stages
  - primarily reducing variance in latter stages

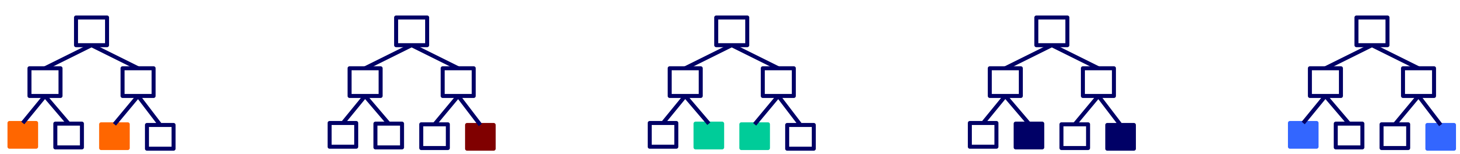
- there is also a margin-maximization interpretation for why boosting works

# Learning models for multi-class problems

- consider a learning task with $k > 2$ classes 
- with some learning methods, we can learn one model to predict the $k$ classes



- an alternative approach is to learn $k$ models; each represents one class vs. the rest



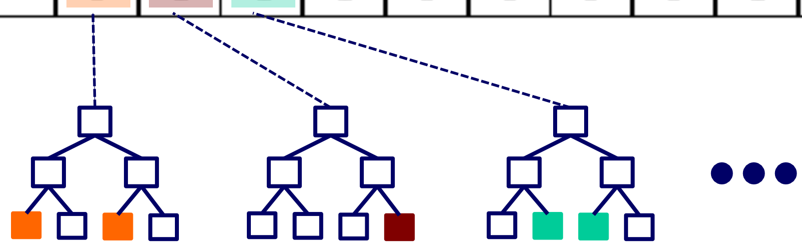
- but we could learn models to represent other encodings as well

# Error correcting output codes

[Dietterich & Bakiri, *JAIR* 1995]

- ensemble method devised specifically for problems with many classes
  - represent each class by a multi-bit code word
  - learn a classifier to represent each bit function

| Class | Code Word | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 6 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 8 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 9 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |



# Classification with ECOC

- to classify a test instance $\boldsymbol{x}$ using an ECOC ensemble with $T$ classifiers
  1. form a vector $h(\boldsymbol{x}) = \langle h_1(\boldsymbol{x}) \ldots h_T(\boldsymbol{x}) \rangle$ where $h_i(\boldsymbol{x})$ is the prediction of the model for the $i^{\text{th}}$ bit
  2. find the codeword $c$ with the smallest Hamming distance to $h(\boldsymbol{x})$
  3. predict the class associated with $c$

- if the minimum Hamming distance between any pair of codewords is $d$, we can still get the right classification with $\left\lfloor \frac{d-1}{2} \right\rfloor$ single-bit errors

recall, $\lfloor x \rfloor$ is the largest integer not greater than $x$

# Error correcting code design

a good ECOC should satisfy two properties

1. *row separation*: each codeword should be well separated in Hamming distance from every other codeword
2. *column separation*: each bit position should be uncorrelated with the other bit positions

| Class | Code Word | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 6 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 8 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 9 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

7 bits apart

6 bits apart

$d = 7$ so this code can correct $\left\lfloor \frac{7-1}{2} \right\rfloor = 3$ errors

# ECOC evaluation with C4.5

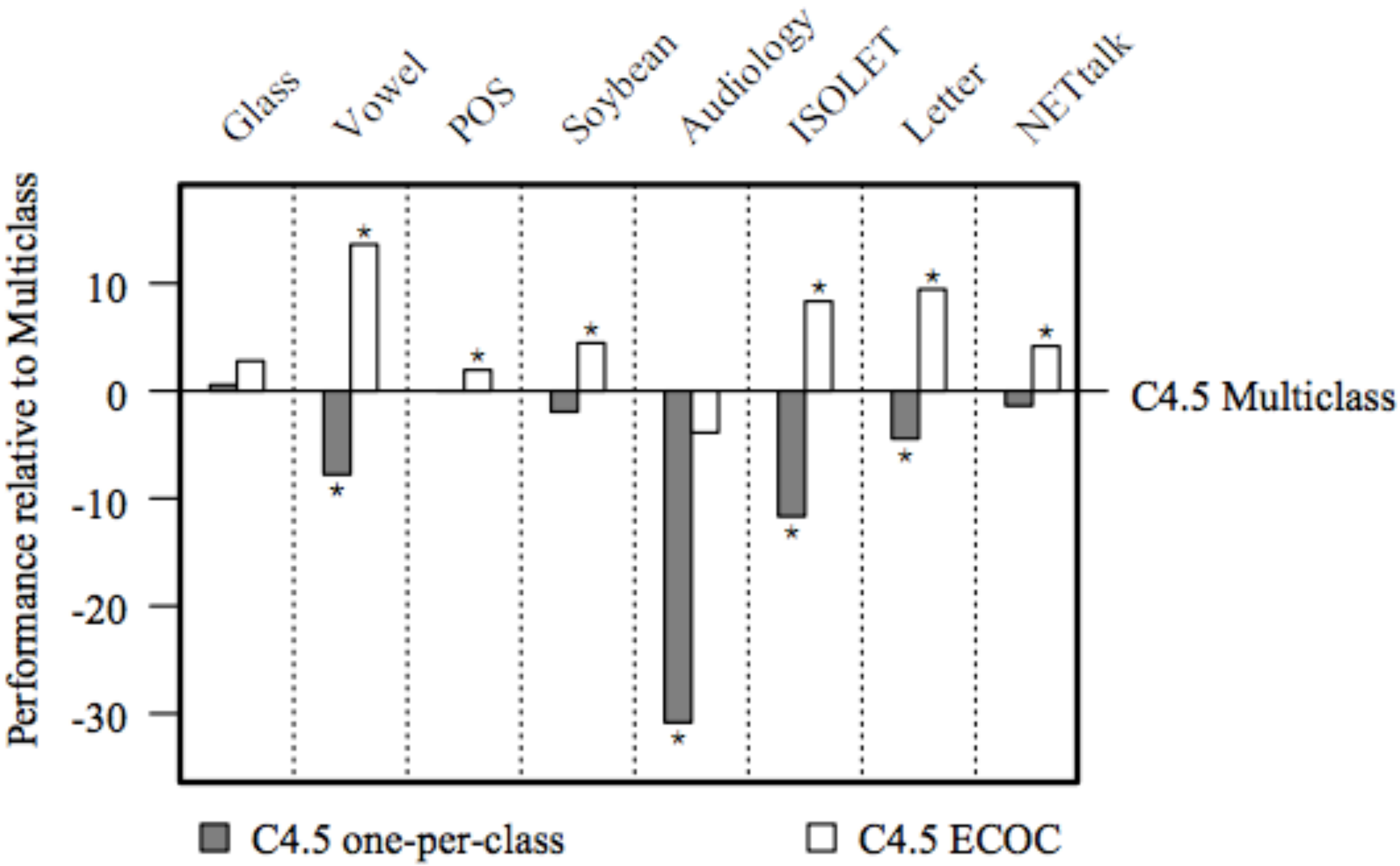



Figure from Bakiri & Dietterich, *JAIR*, 1995

# ECOC evaluation with neural nets

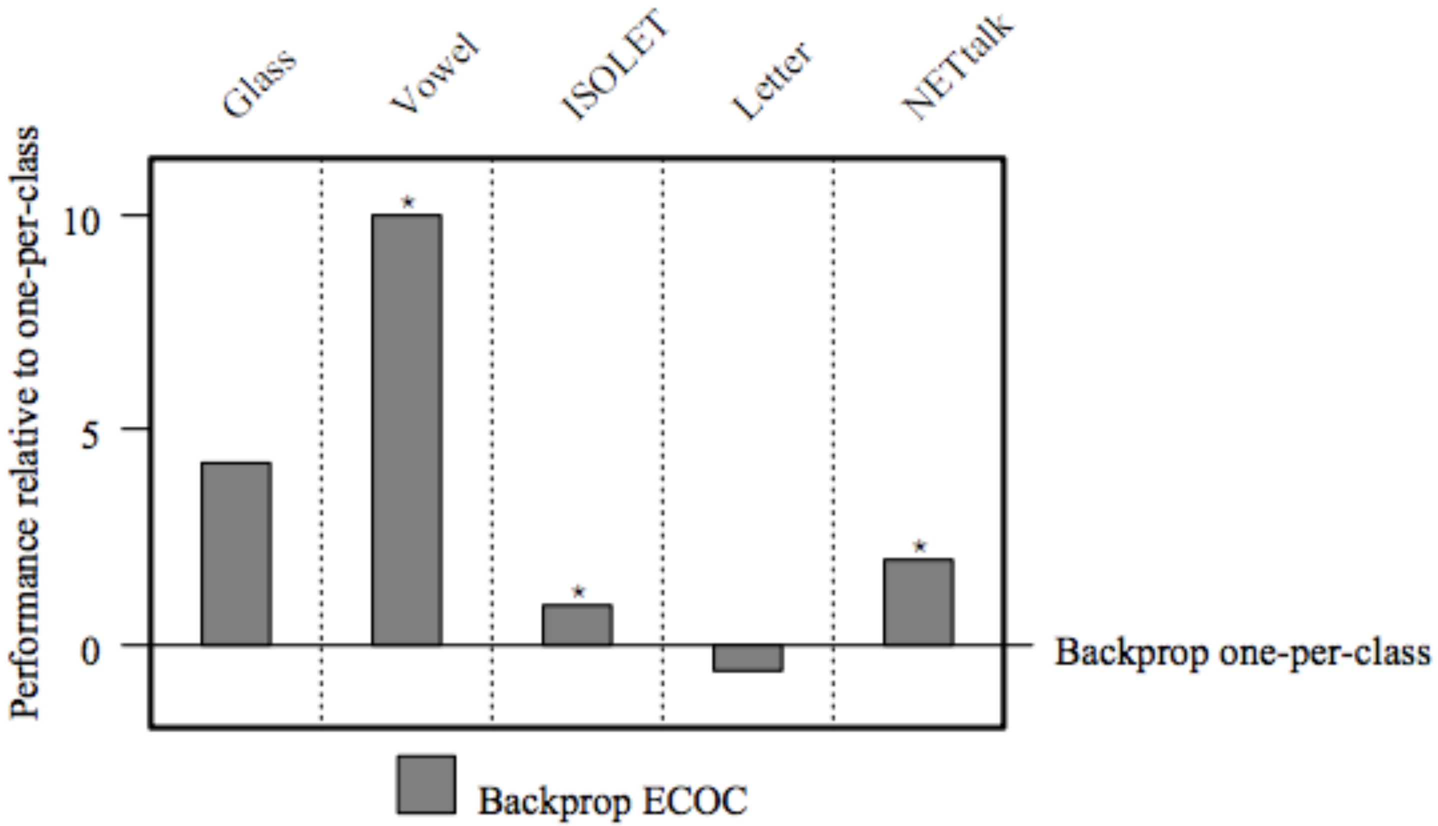



Figure from Bakiri & Dietterich, *JAIR*, 1995

# (Functional) Gradient Boosting

- Consider learning a regression tree to minimize squared error
- Boosting adds a new tree (or model of any base learner type) to fix current errors, by reweighting wrongly-predicted examples; Breiman realized could just fit next tree to current residuals
  - Current model: $F(\mathbf{x}) = w_1 F_1(\mathbf{x}) + \ldots + w_n F_n(\mathbf{x})$
  - Each example $(\mathbf{x}_i, y_i)$ *now* becomes $(\mathbf{x}_i, r_i)$, where $r_i = y_i - F(\mathbf{x}_i)$
- Friedman, Bartlett, others saw residual $y_i - F(\mathbf{x}_i)$ is just gradient of squared error loss $\frac{1}{2}(y_i - F(\mathbf{x}_i))^2$ with respect to $F(\mathbf{x}_i)$; in general, can fit next model to negative gradient of any loss function if can efficiently find a model aligned with negative gradient of that loss

# Gradient Boosting with Squared Error (from Friedman, 1999)

$F_0(\mathbf{x}) = \bar{y}$

For $m = 1$ to $M$ do:

$$\tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i), \quad i = 1, N$$

$$(\rho_m, \mathbf{a}_m) = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^{N} [\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{a})]^2$$

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$$

endFor

# TreeBoost (Friedman)

- Friedman, Hastie, collaborators realized that once you learned the next tree, instead of fitting one best coefficient (weight) to the tree, why not re-fit a whole vector of coefficients, one per leaf

- This is tree boost, algorithm on next slide (slide from Hastie, 1999)

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.

2. For $m = 1$ to $M$:

   (a) For $i = 1, 2, \dots, N$ compute

   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm},\ j = 1, 2, \dots, J_m$.

   (c) For $j = 1, 2, \dots, J_m$ compute

   $$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

3. Output $\hat{f}(x) = f_M(x)$.

# Other Ensemble Methods

- Use different parameter settings with same algorithm
- Use different learning algorithms
- Instead of voting or weighted voting, learn the combining function itself
  - Called "Stacking"
  - Higher risk of overfitting
  - Ideally, train arbitrator function on different subset of data than used for input models
- Naïve Bayes is weighted vote of stumps

# Comments on ensembles

- They very often provide a boost in accuracy over base learner
- It's a good idea to evaluate an ensemble approach for almost any practical learning problem
- They increase runtime over base learner, but compute cycles are usually much cheaper than training instances
- Some ensemble approaches (e.g. bagging, random forests) are easily parallelized
- Prediction contests (e.g. Kaggle, Netflix Prize) usually won by ensemble solutions
- Ensemble models are usually low on the comprehensibility scale, although see work by

  [Craven & Shavlik, *NIPS* 1996]

  [Domingos, *Intelligent Data Analysis* 1998]

  [Van Assche & Blockeel, *ECML* 2007]