Ensemble Methods

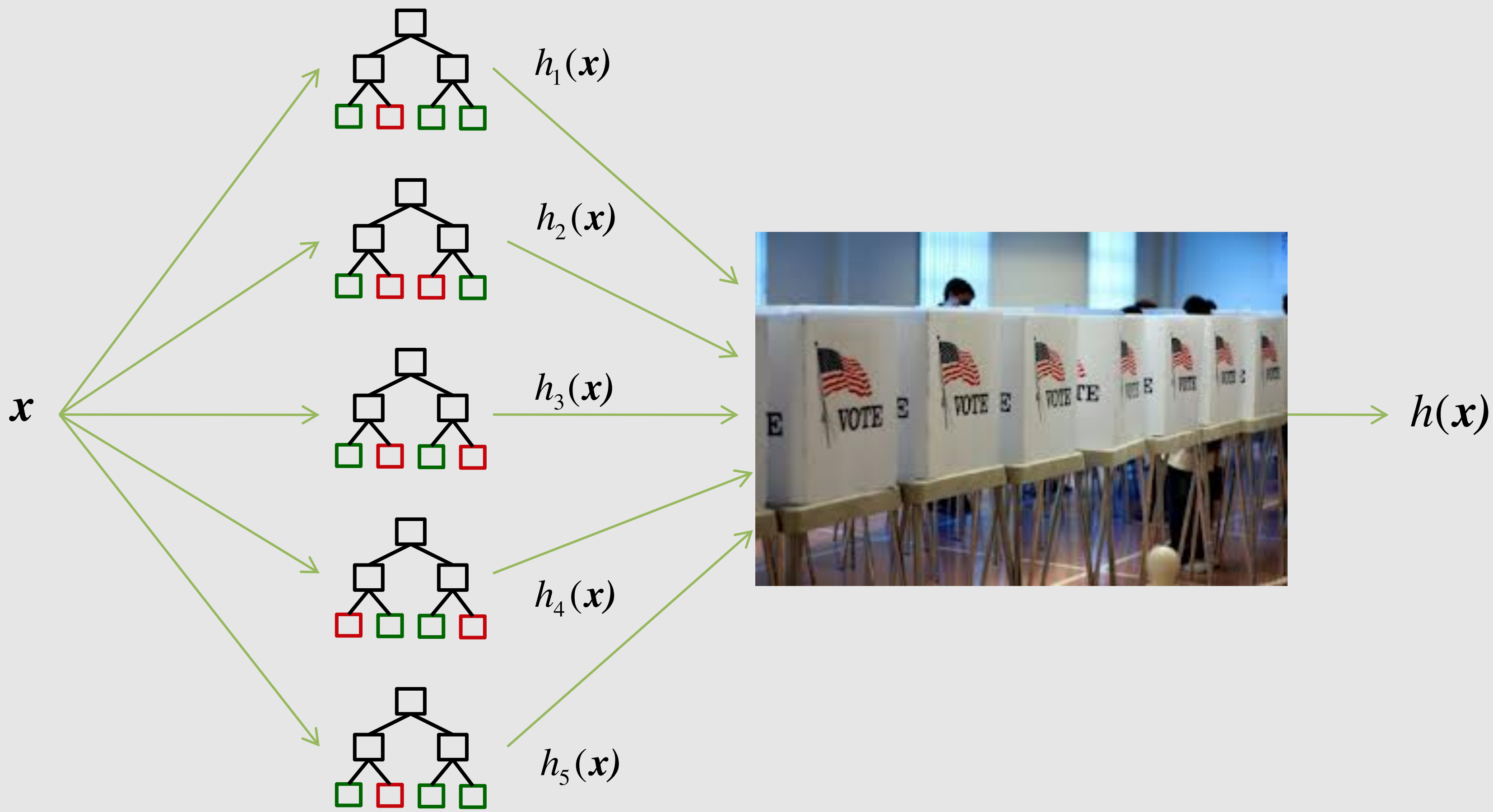
CS 760@UW-Madison

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…

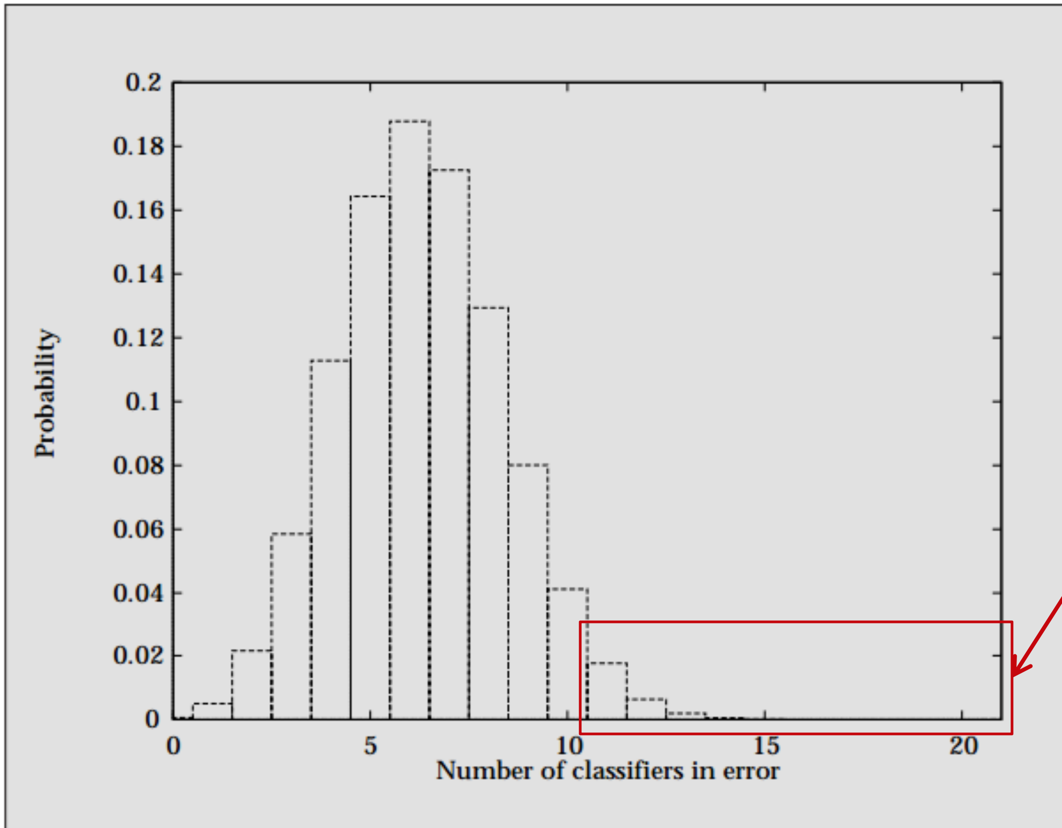


Figure 1. The Probability That Exactly ℓ (of 21) Hypotheses Will Make an Error, Assuming Each Hypothesis Has an Error Rate of 0.3 and Makes Its Errors Independently of the Other Hypotheses.

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, skewing*)
  - choosing different features and subsamples (*random forests*)

Bagging (Bootstrap Aggregation)
[Breiman, *Machine Learning* 1996]

learning:
given: learner L, training set $D = \{ \langle \boldsymbol{x}_1, y_1 \rangle \ \dots \ \langle \boldsymbol{x}_m, y_m \rangle \}$
for $i \leftarrow 1$ to T do
    $D^{(i)} \leftarrow m$ instances randomly drawn with replacement from D
    $h_i \leftarrow$ model learned using L on $D^{(i)}$

classification:
given: test instance $\boldsymbol{x}$
predict $y \leftarrow$ plurality_vote($h_1(\boldsymbol{x}) \dots h_T(\boldsymbol{x})$)

regression:
given: test instance $\boldsymbol{x}_t$
predict $y \leftarrow$ mean($h_1(\boldsymbol{x}) \dots h_T(\boldsymbol{x})$)

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, i.e., those that tend to *overfit* training data

Empirical evaluation of bagging with C4.5

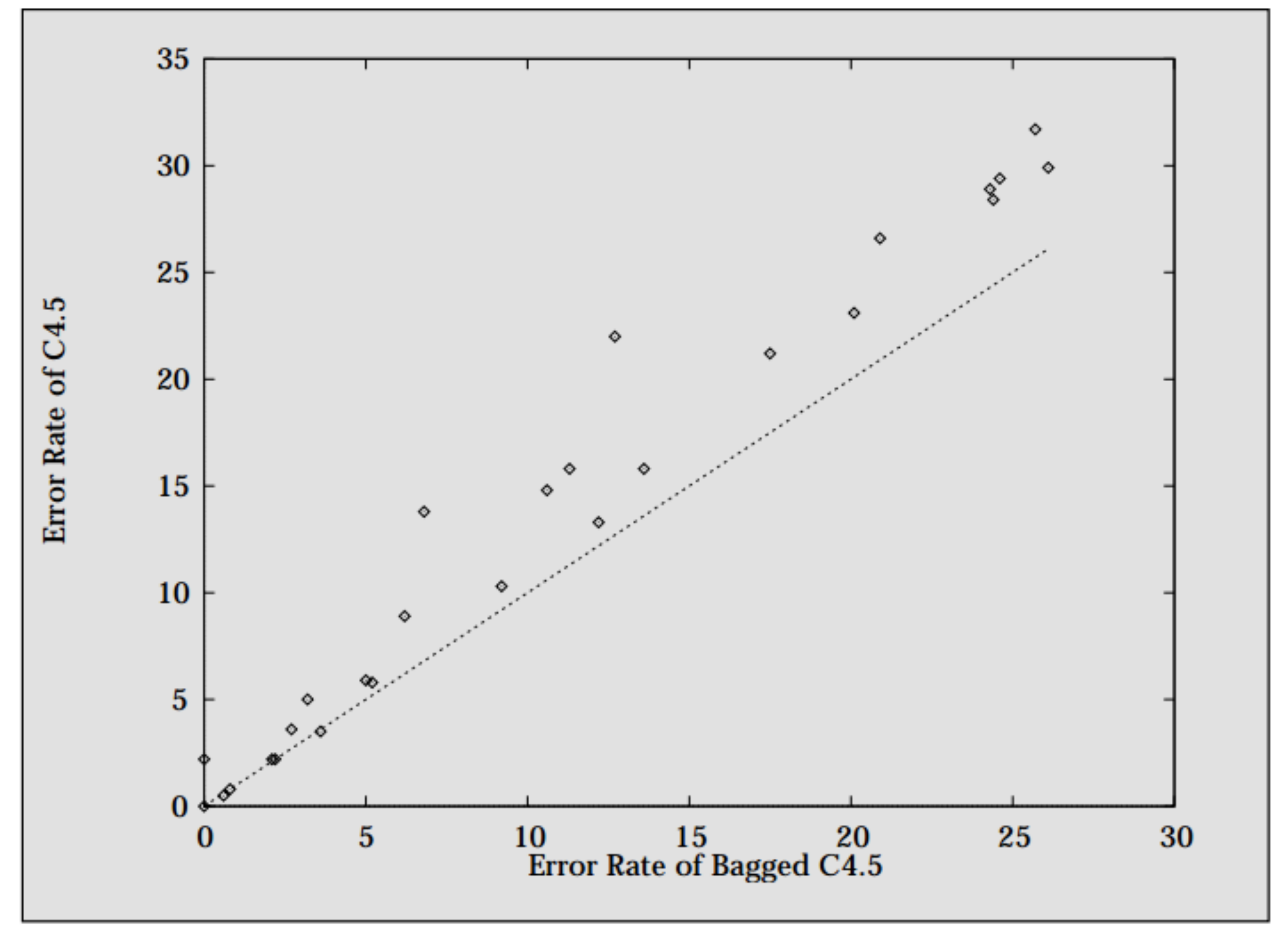


Figure from Dietterich, *AI Magazine*, 1997

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

Boosting

- Boosting came out of the PAC learning community

- A *weak PAC learning* algorithm is one that cannot PAC learn for arbitrary ε and δ, but it can for some: its hypotheses are at least 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 : $p_t(i) \leftarrow w_t(i) \,/\, \left(\sum_j w_t(j)\right)$ // 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

  $\beta_t \leftarrow \varepsilon_t \,/\, (1 - \varepsilon_t)$ // lower error, smaller β_t

  for all i where $h_t(\boldsymbol{x}_i) = y_i$ // downweight correct examples

    $w_{t+1}(i) \leftarrow w_t(i)\, \beta_t$

return:

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

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. Sample a large ($\gg m$) unweighted set of instances according to p_t; run L in the ordinary manner

Empirical evaluation of boosting with C4.5

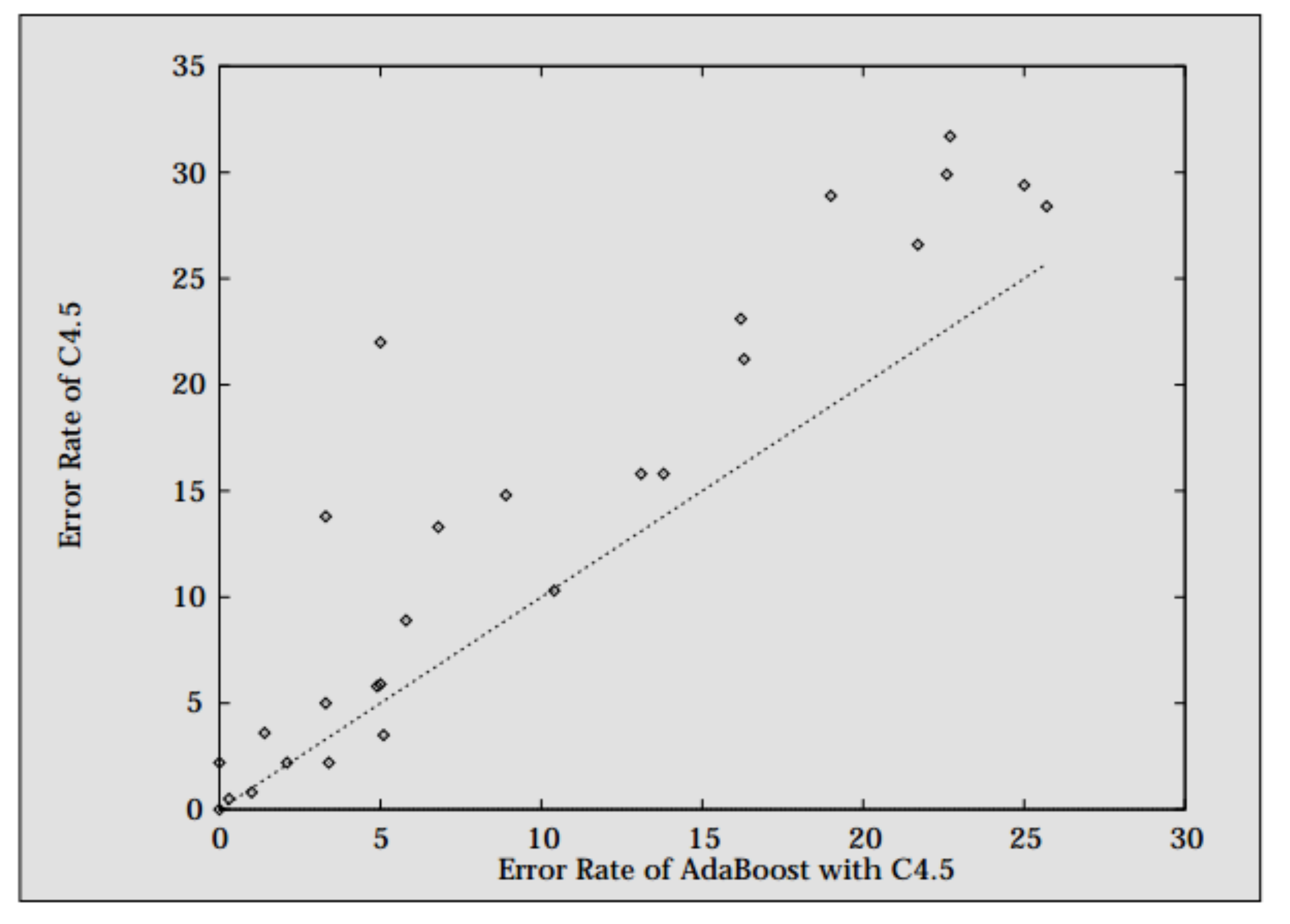


Figure from Dietterich, *AI Magazine*, 1997

Bagging and boosting with C4.5

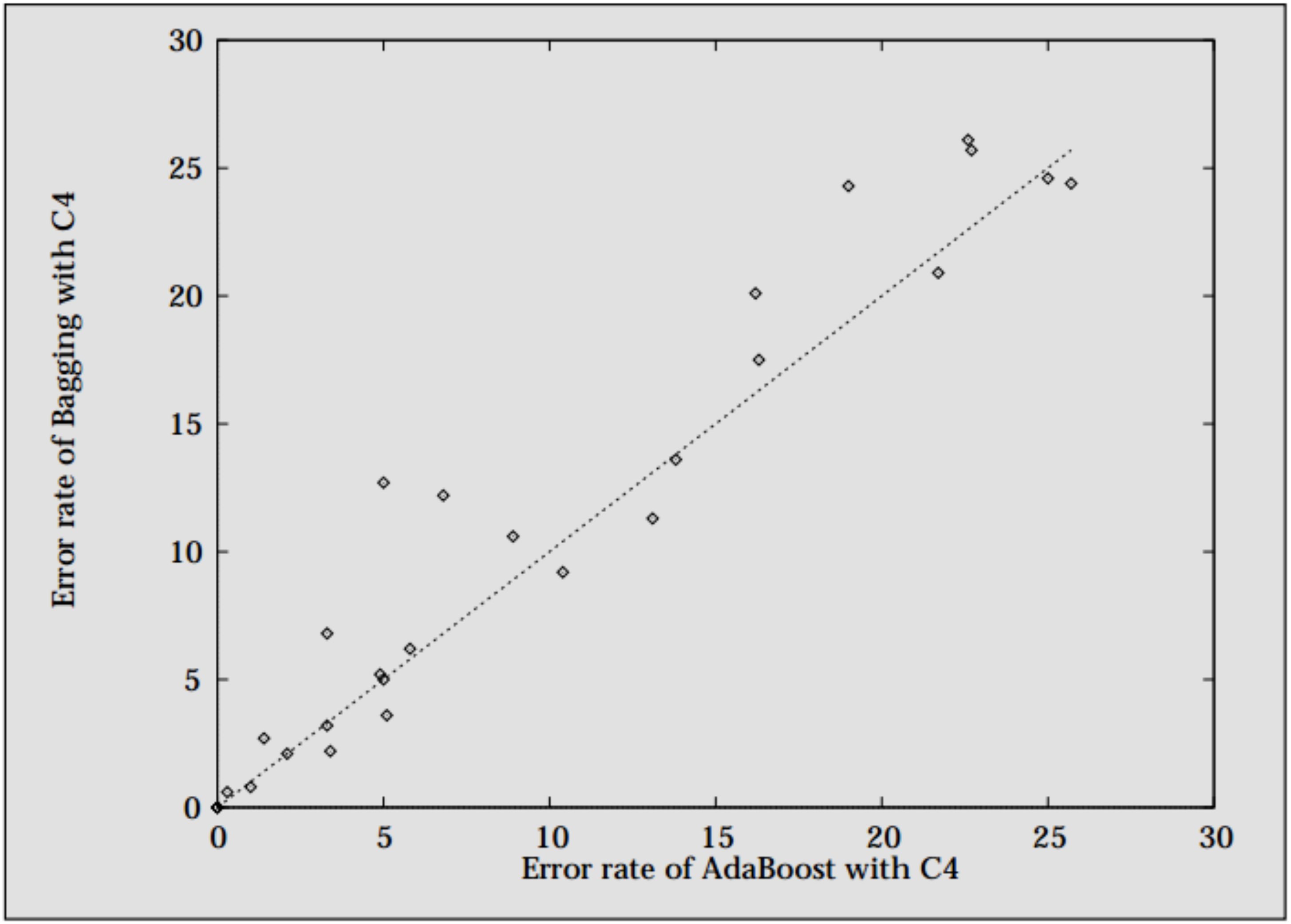


Figure from Dietterich, *AI Magazine*, 1997

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]

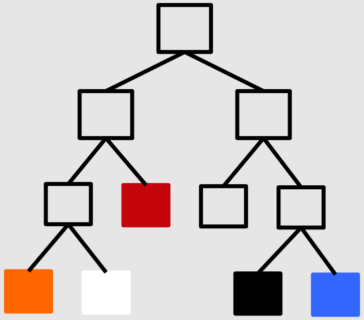
given: candidate feature splits F, training set $D = \{ \langle \boldsymbol{x}_1, y_1 \rangle \ \ldots \ \langle \boldsymbol{x}_m, y_m \rangle \}$
for $i \leftarrow 1$ to T do
$D^{(i)} \leftarrow m$ instances randomly drawn with replacement from D
$h_i \leftarrow$ randomized decision tree learned with 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 $f \approx \sqrt{|F|}$)
choose the best feature split in R
do not prune trees

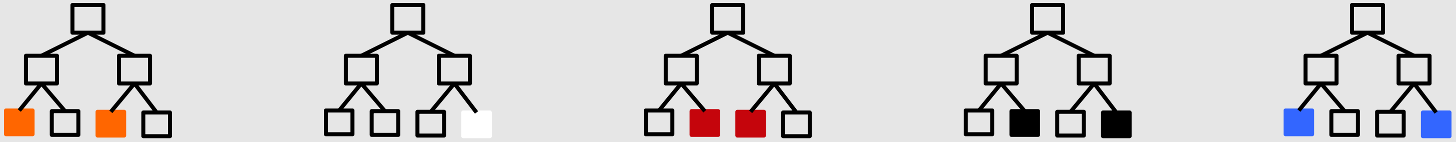
classification/regression:
as in bagging

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



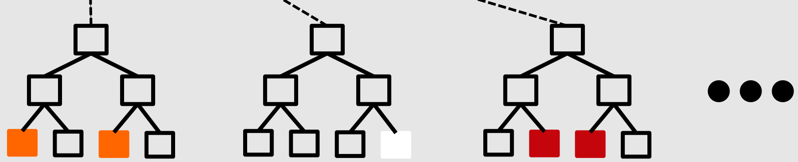
- but we could learn models to represent other encodings as well

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^{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

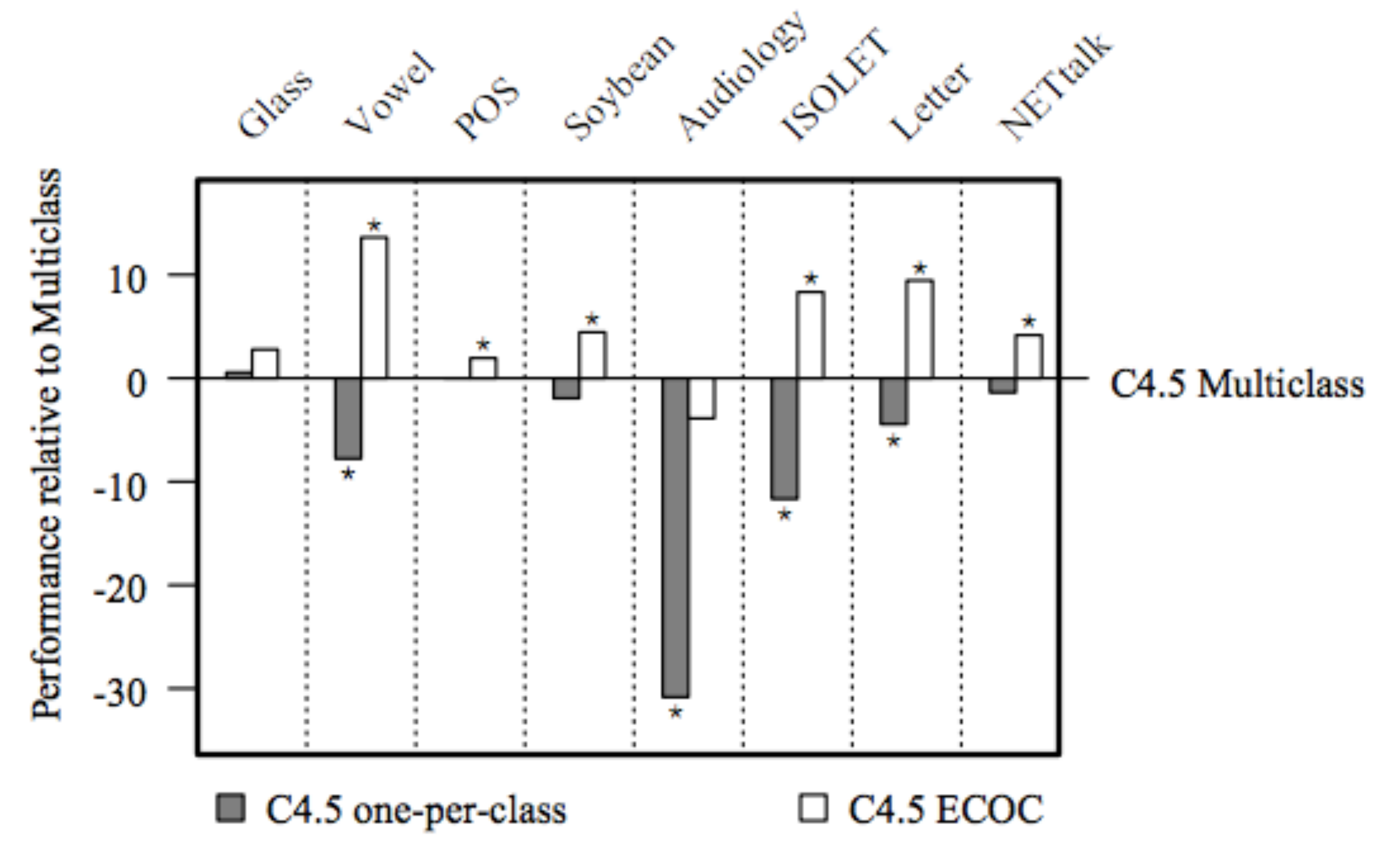


Figure from Bakiri & Dietterich, *JAIR*, 1995

ECOC evaluation with neural nets

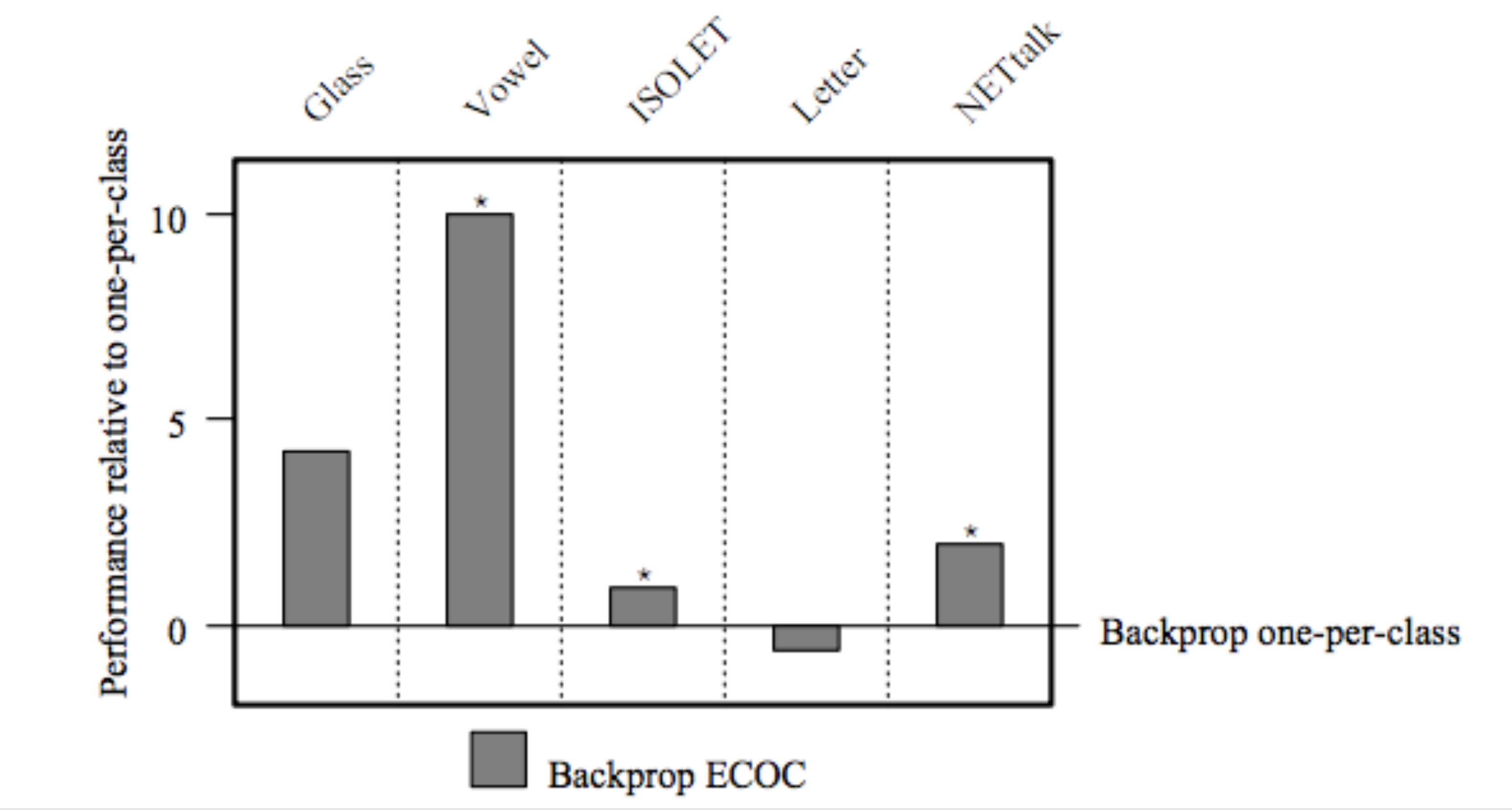


Figure from Bakiri & Dietterich, *JAIR*, 1995

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]

THANK YOU

Some of the slides in these lectures have been adapted/borrowed from materials developed by Yingyu Liang, Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, and Pedro Domingos.