Machine Learning

Prof. Dr. Stefan Kramer
Johannes Gutenberg-Universität Mainz

Outline

- Bagging and Random Forests
- Boosting
- Bias-Variance Decomposition

Bagging and Random Forests

Bagging

- **B**ootstrap **agg**regat**ing**
- Simplest way of combining predictions: voting/averaging, where each model receives equal weight
- How can we vote if we have only one dataset?

Bootstrap

- Resampling with replacement
- Repeatedly: Given dataset of size n, sample new dataset of size n by drawing with replacement (also called 0.632 bootstrap)
- A particular instance has a probability of 1-1/n of not being picked
- Thus its probability of ending up in the test data is: $$\left(1-\frac{1}{n}\right)^n \approx e^{-1} = 0.368$$
- Thus, a bootstrap sample will contain approx. 63.2% of the instances

More on Bagging

- Bagging reduces the variance component of the expected error by voting/averaging
- Improves performance almost always if the base classifier is *unstable* and the data are *noisy*

Bagging Classifiers

model generation

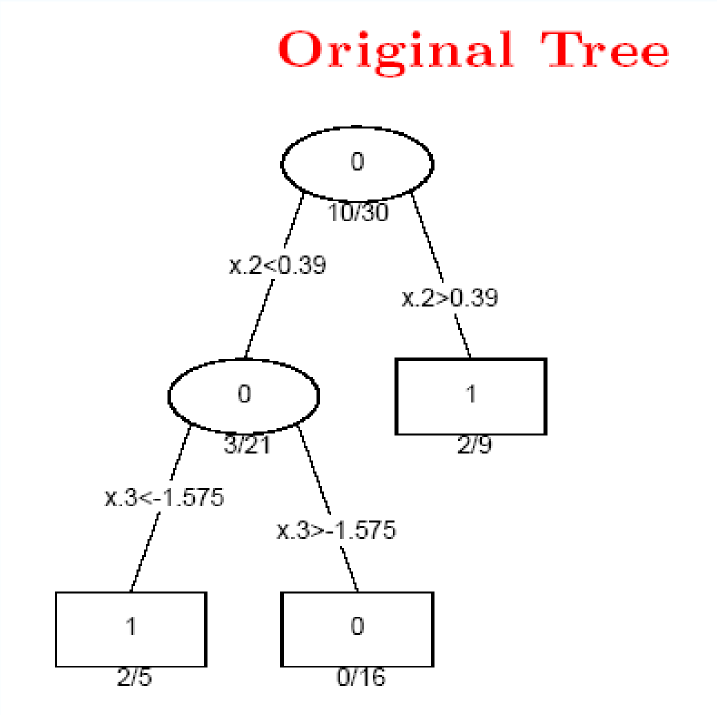
```
Let n be the number of instances in the training data.
For each of t iterations:
  Sample n instances with replacement from training set.
  Apply the learning algorithm to the sample.
  Store the resulting model.
```

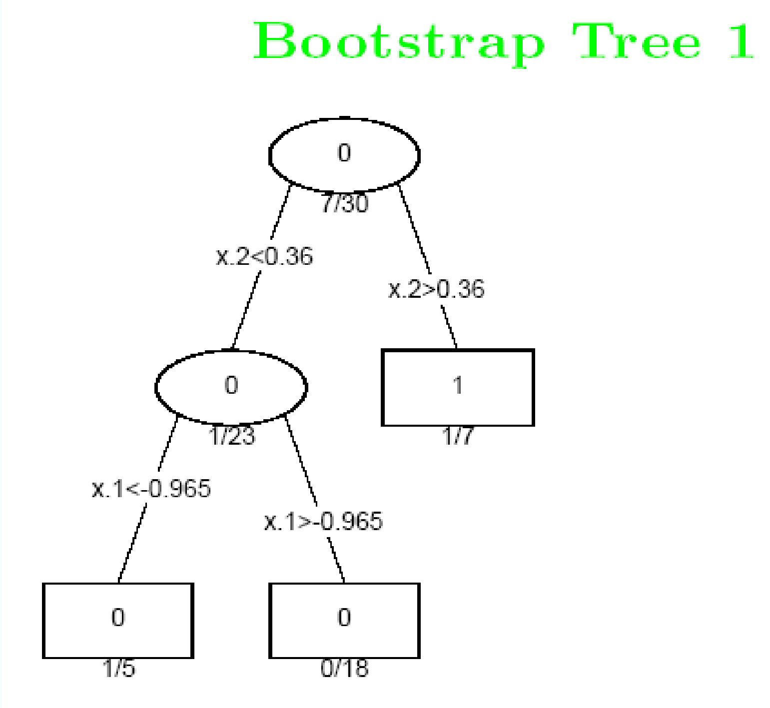
classification

```
For each of the t models:
  Predict class of instance using model.
Return class that has been predicted most often.
```

Example

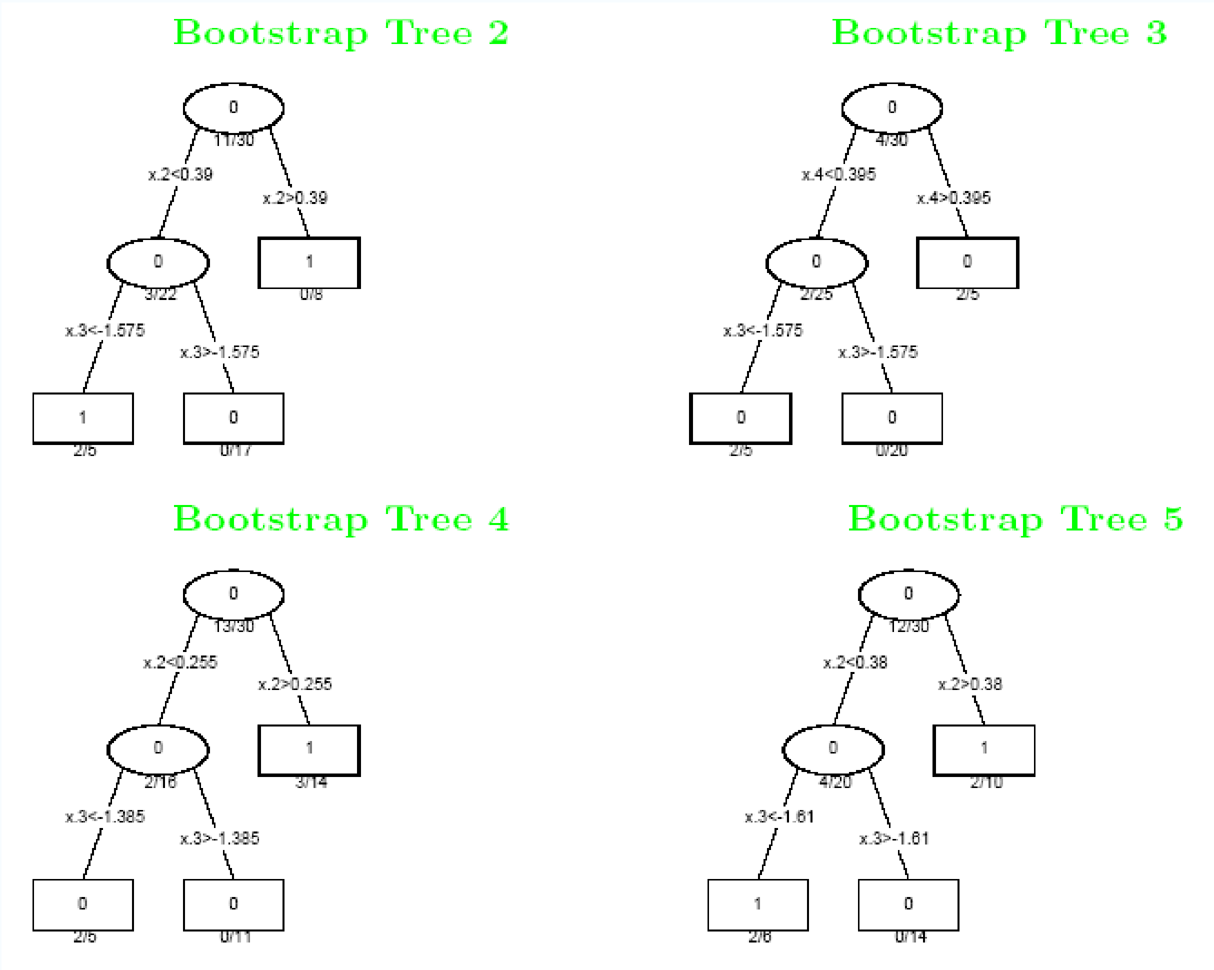
Tree with simulated data





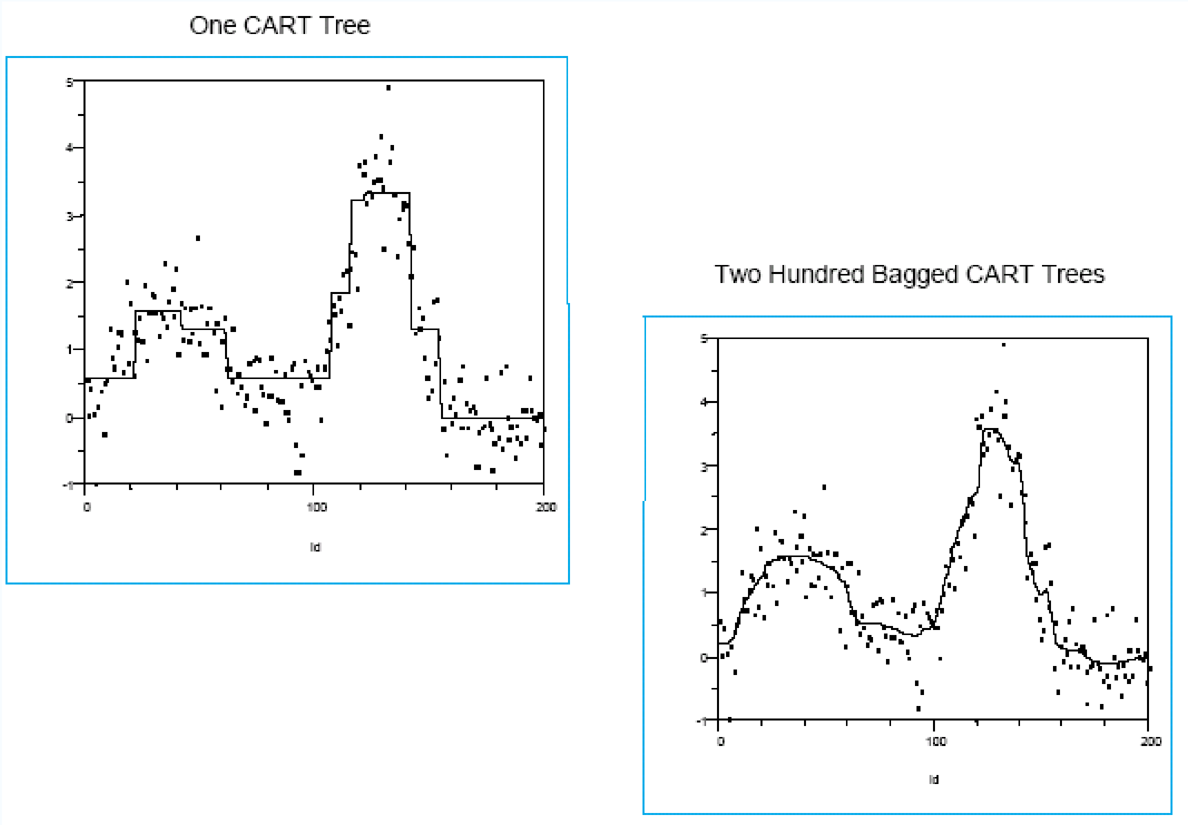
Elements of Statistical Learning (c) Hastie, Tibshirani & Friedman 2001

Example


Bootstrap Tree 2
0
11/30
x.2<0.39
x.2>0.39
0
3/22
1
0/8
x.3<-1.575
x.3>-1.575
1
2/5
0
0/17
Bootstrap Tree 3
0
4/30
x.4<0.395
x.4>0.395
0
2/25
0
2/5
x.3<-1.575
x.3>-1.575
0
2/5
0
0/20
Bootstrap Tree 4
0
13/30
x.2<0.255
x.2>0.255
0
2/16
1
3/14
x.3<-1.385
x.3>-1.385
0
2/5
0
0/11
Bootstrap Tree 5
0
12/30
x.2<0.38
x.2>0.38
0
4/20
1
2/10
x.3<-1.61
x.3>-1.61
1
2/6
0
0/14

Example

CART (Classification And Regression Trees)



Rick Higgs & Dave Cummins, Statistical & Information Sciences, Lilly Research Laboratories, 2003

Random Forests

- Due to Breiman (2001)
- Combines his bagging idea with random feature selection
- Parameters:
  - n, the number of instances
  - m, the number of features
  - k, the number of randomly selected features
  - l, the number of iterations

Training RandomForest(D_{Trg}, k)

for i := 1 to I

  (D'_{Trg}, D'_{Tst}) :=

    CreateBootstrapSample(D_{Trg})

  % sample n times with replacement

  t_i := fully grown (i.e. unpruned) decision

    tree; for each node, randomly sample

    k variables from which the best is chosen

    based on D'_{Trg}

   % many possible uses of D'_{Tst}: estimate out-of-

    % bag error, evaluate features, etc.

Random Forests

- Testing (prediction): majority vote among the ensemble members

- Currently, next to support vector machines (SVMs), among the best performing simpler machine learning algorithms, but typically faster than (non-linear) SVMs
- Question of performance on high-dimensional data

Boosting

Boosting

- Stems from computational learning theory
  (weak and strong PAC learning)
- Weighted voting
- Not parallelizable:
  sequence of learning tasks
- New model is encouraged to become expert for instances incorrectly classified by previous models

PAC Learning

- PAC: Probably Approximately Correct
- Concept class C, hypothesis language H, instance space X, learning algorithm L
- C is PAC-learnable by L using H, if,
  - for any $c \in C$
  - for any distribution D over X
  - $0 < \varepsilon < 0.5$
  - $0 < \delta < 0.5$
- …

PAC Learning

- …
- Algorithm L will output with probability $1-\delta$ a hypothesis h with $error_D(h) < \varepsilon$ in time that is polynomial in $1/\varepsilon$, $1/\delta$, $|X|$ and size(c).
- *Learning algorithm has to achieve arbitrarily small values for ε and δ, and the runtime is polynomially bounded by $1/\varepsilon$ and $1/\delta$*

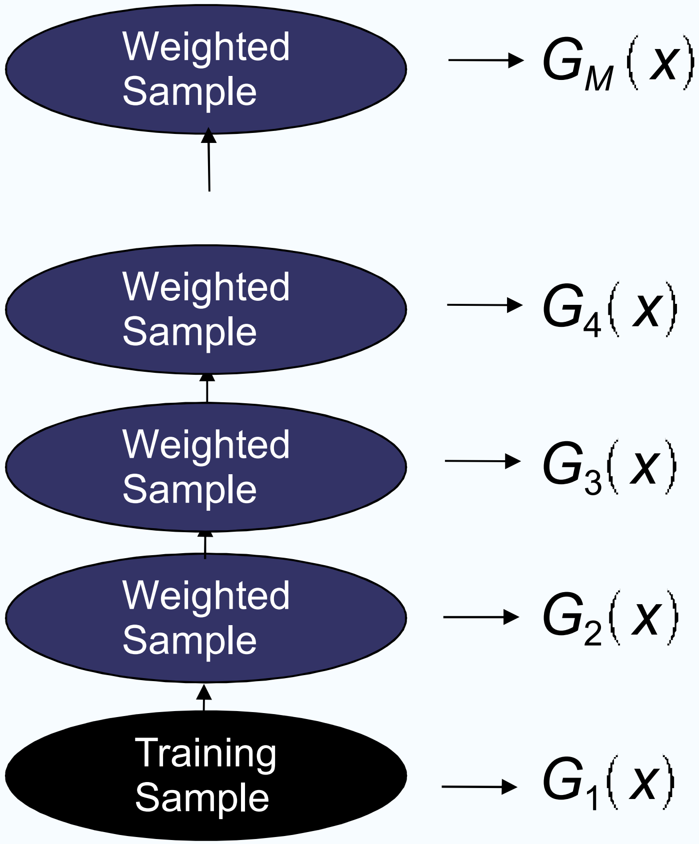
Note

- Why distinct C and H?
- Can be identical
- But definition with C and H also allows statements like:
  L *learns k-term DNF (maximally* k *disjunctions) via k-CNF (maximally* k *variables in each disjunction)*

The Boosting Problem

- “strong“ PAC algorithm
  - for any distribution
  - $\forall\ \varepsilon > 0$ and $\delta > 0$
  - given polynomially many random examples
  - finds hypothesis with error $\leq \varepsilon$
    with probability $\geq 1- \delta$
- “weak“ PAC algorithm
  - same, but only for error < 0.5
    (just a bit better than random guessing)
- What is the relationship between strong and weak PAC learning?

AdaBoost.M1



Two-class problem $Y \in \{-1; 1\}$

G() produces a prediction from a vector of variables X

Final prediction obtained by using a weighted vote

$$G(x) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Discrete AdaBoost

initialize the observation weights (set w_i to 1/N)

for m=1 **to** M **do**

induce a classifier $G_m(x)$ to the training data using weights w_i

$$err_m = \sum_{J=1}^{N} w_j \times I(y_j \neq G_m(x_j))$$

$$\alpha_m = \frac{1}{2}\log(\frac{1-err_m}{err_m})$$

$$0 < err_m < 0.5$$

for j=1 **to** N **do**

if $y_j \neq G(x_j)$ **then**

$$w_j \leftarrow w_j \cdot (1-err_m)/err_m$$

renormalize such that $\sum_{j=1}^{N} w_j = 1$

output $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$

Discrete AdaBoost

initialize the observation weights (set w_i to 1/N)
for m=1 **to** M **do**
induce a classifier $G_m(x)$ to the training data using weights w_i

$$err_m = \sum_{J=1}^{N} w_j \times I(y_j \neq G_m(x_j))$$

$$\alpha_m = \frac{1}{2}\log(\frac{1-err_m}{err_m})$$

$0<err_m<0.5$

for j=1 **to** N **do**
if $y_j = G(x_j)$ **then**

$$w_j \leftarrow w_j \cdot err_m /(1-err_m)$$

renormalize such that $\sum_{j=1}^{N} w_j = 1$

output $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$

Adaboost with Confidence Weighted Predictions (RealAB)

initialize the observation weights (set w_i to 1/N)
for m=1 **to** M **do**
induce a classifier $G_m(x)$ to the training data using weights w_i

$G_m \in [-1,+1]$

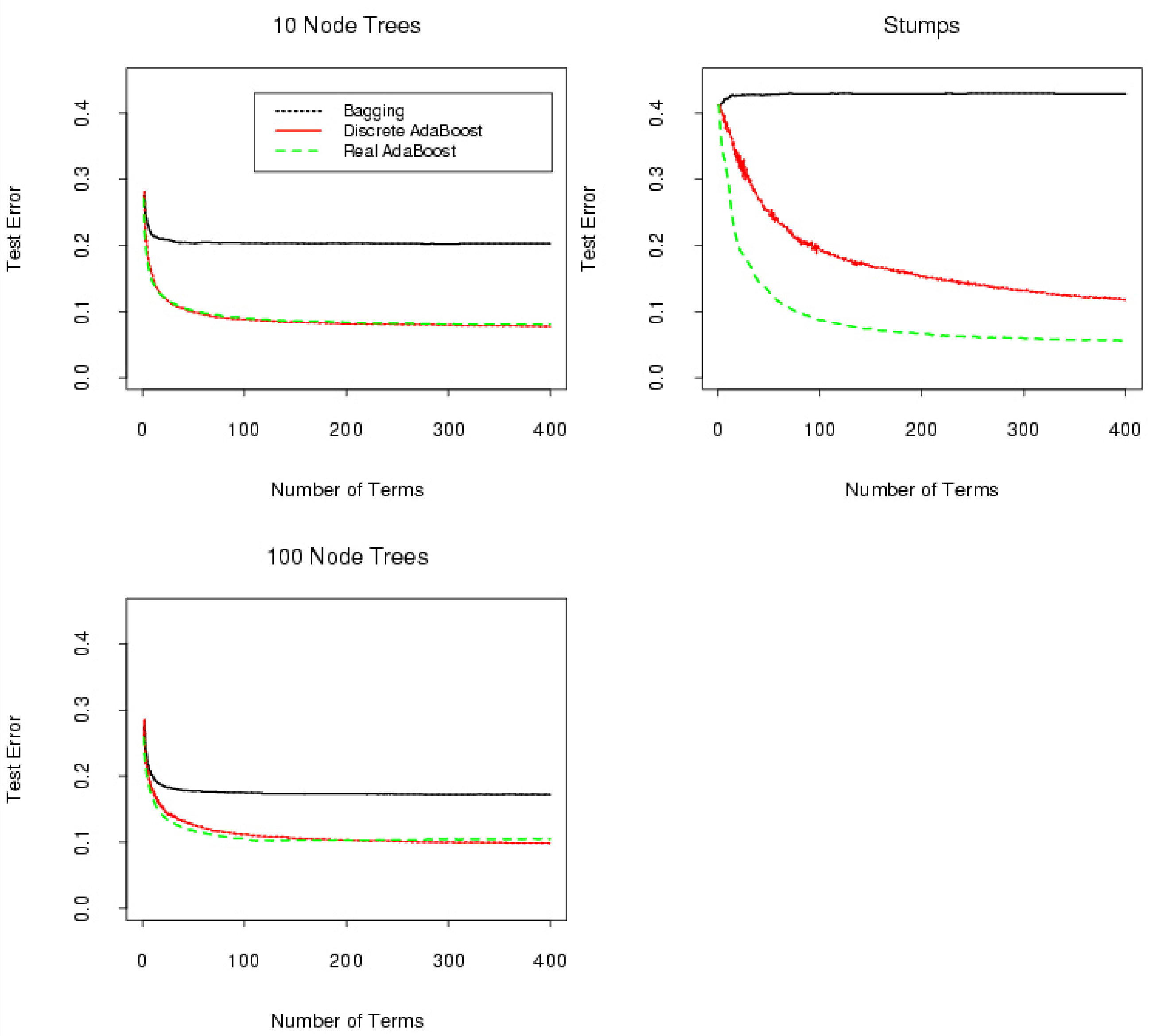
find $\alpha_m \in \Re$

$w_j \leftarrow w_j \cdot \exp(-\alpha_m y_j G_m(x_j)) / Z_m$

$0 < err_m < 0.5$

output $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$

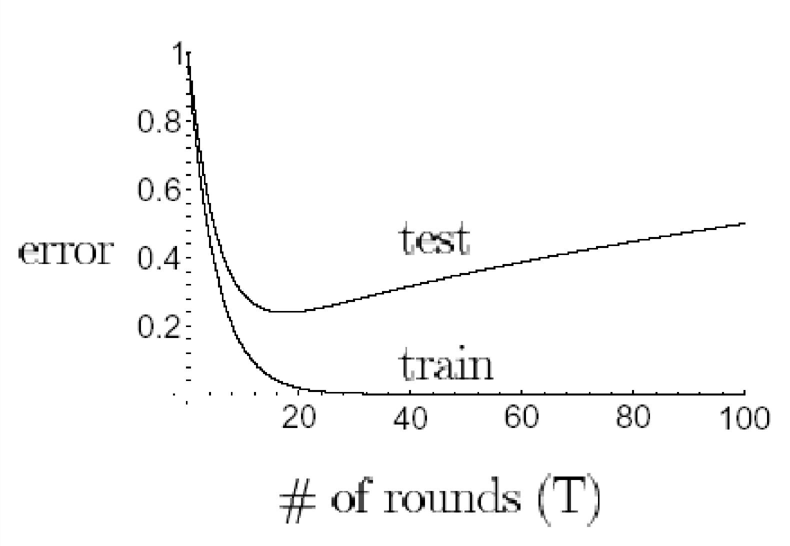
Comparison

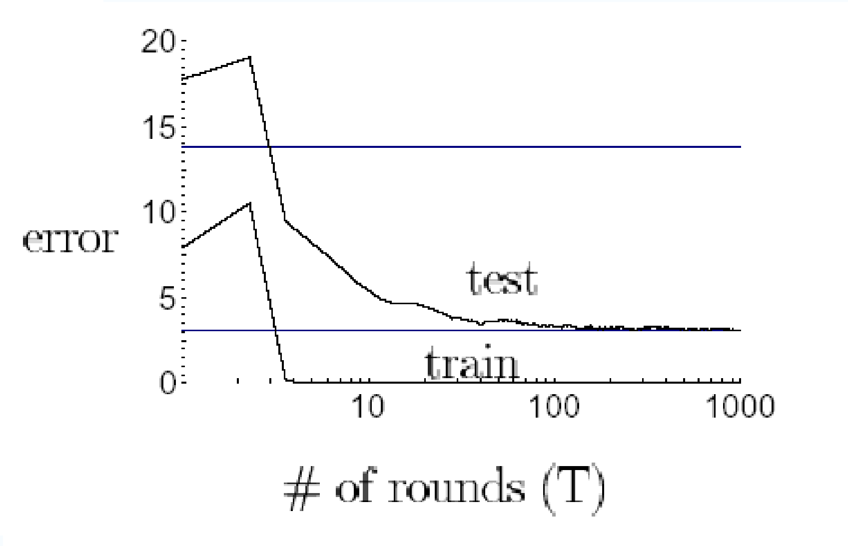

10 Node Trees
Stumps
100 Node Trees
Bagging
Discrete AdaBoost
Real AdaBoost
Test Error
Number of Terms
0.0
0.1
0.2
0.3
0.4
0
100
200
300
400

Notes

- Theory: training error decreases exponentially
- Boosting works with *weak learners*:
  only condition is that error *does not exceed 0.5*
- Base classifier should be able to handle weighted instances
- However, can be applied without weights using resampling with probability determined by weights:
  - disadvantage: not all instances are used
  - advantage: resampling can be repeated if error exceeds 0.5

Observation with AdaBoost





Expected learning curve

Observed learning curve

What's going on? (Occam's Razor)

Thesis

- Paradox disappears if we consider the „confidence“ in the prediction, the so-called *margin*:

$$G(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

$$G_{final}(x) = sign(G(x))$$

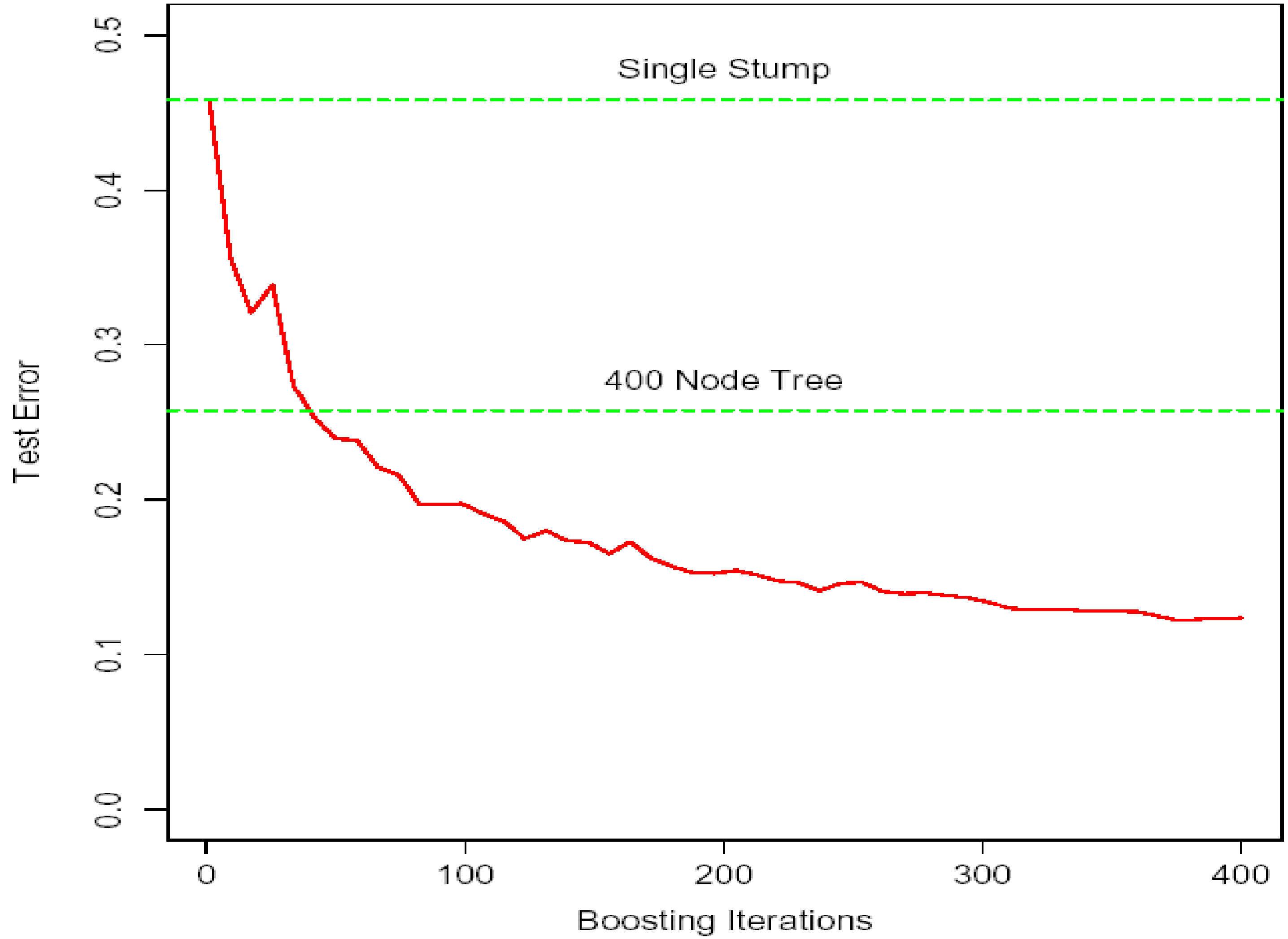
$$m_G(x, y) = y \cdot G(x)$$

- „Larger margins on the training set yield smaller generalization errors.“

Boosting

- Empirical results:
  boosting can reduce the error dramatically, helps almost always
- (Quinlan 96): sometimes it is fooled by noise in the data and not as stable as bagging (though bagging performs not as well on average)
- Boosted learning algorithms have quite a low bias

Example: Performance Boosting



Bias-Variance Decomposition

Introduction

- *Why do ensemble methods work?*
- Decomposition of error into so-called *bias* and *variance* term
- *Bagging* reduces *variance*
- *Boosting* usually (with simple models) *first* reduces *bias*, *then variance*

Bias and Variance

- *Bias* measures how close the average classifier produced by the learning algorithm will be to the target function (kind of *systematic error*)
- *Variance* measures how much each of the learning algorithm's guesses will vary with respect to each other /
  how the error *depends on a particular training set*

Bias/Variance Decomposition

- Assume we have an infinite number of classifiers built from different training sets of size n
- *Bias*: expected error of the combined model on new data
- *Variance*: expected error of a learning scheme due to a particular training set
- Total expected error = bias + variance + irreducible error

Bias-Variance Decomposition for Regression

Assume $Y = f(X) + \varepsilon$, where $E(\varepsilon) = 0$, then the expression for the expected *prediction error* of a regression fit with squared-error loss is

$$PE(f(.,T)) = E_{X,Y}(Y - f(X,T))^2$$

$$PE^*(f) = \mathrm{E}_T[PE(f(.,\mathrm{T}))] =$$

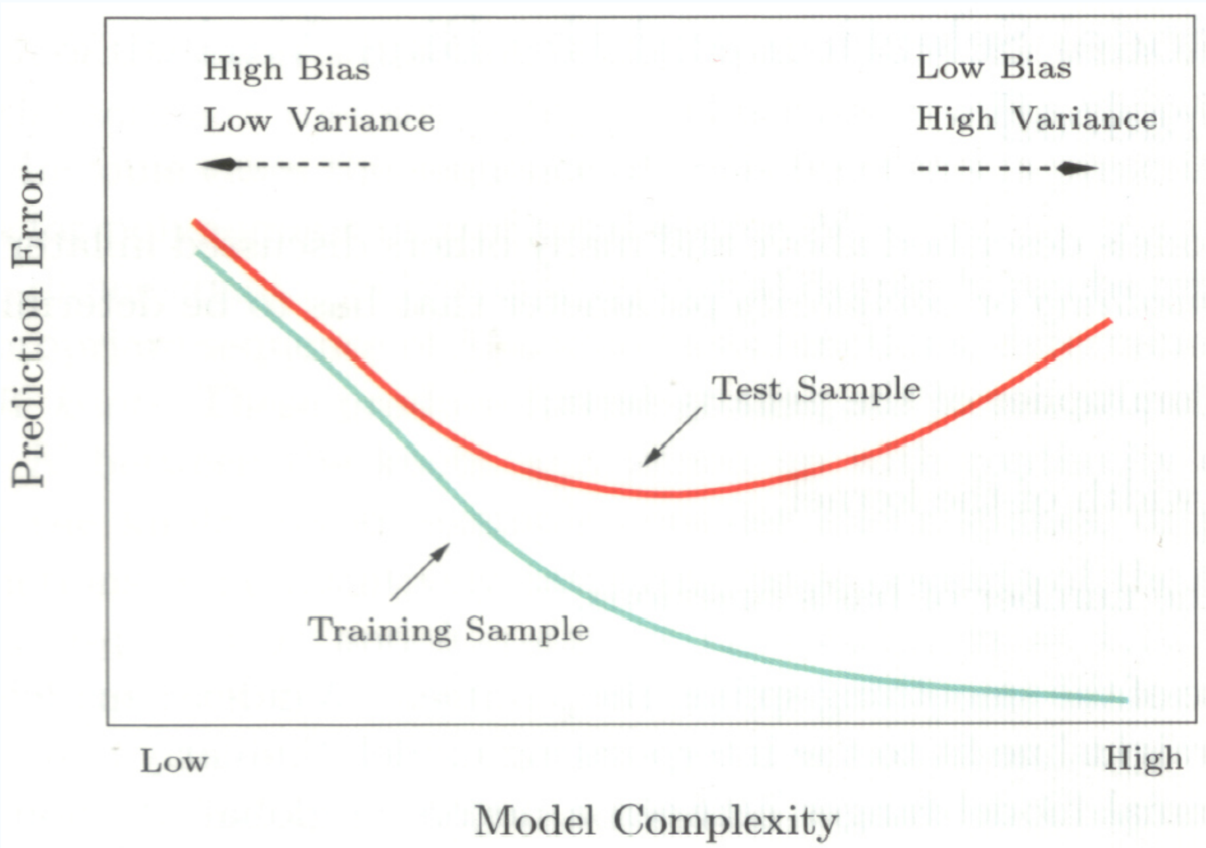
$$= E\varepsilon^2 + E_X[f^*(X) - \hat{f}(X)]^2 + E_{X,T}[f(X,T) - \hat{f}(X)]^2$$

$$= E\varepsilon^2 + \mathrm{Bias}(\hat{f}(X))^2 + \mathrm{Var}(\hat{f}(X))$$

$$= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}$$

T is a randomly drawn training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$

Bias and Variance



Behavior of test sample and training sample error as the model complexity is varied

Conclusion

- Ensembles: mostly motivated by theory
- Comprehensibility suffers
- Standard methods for improving performance of relatively simple basic models