Ensemble Learning

COMP9417 Machine Learning and Data Mining

May 9, 2017

Acknowledgements

Material derived from slides for the book
"Elements of Statistical Learning (2nd Ed.)" by T. Hastie,
R. Tibshirani & J. Friedman. Springer (2009)
http://statweb.stanford.edu/~tibs/ElemStatLearn/

Material derived from slides for the book
"Machine Learning: A Probabilistic Perspective" by P. Murphy
MIT Press (2012)
http://www.cs.ubc.ca/~murphyk/MLbook

Material derived from slides for the book
"Machine Learning" by P. Flach
Cambridge University Press (2012)
http://cs.bris.ac.uk/~flach/mlbook

Material derived from slides for the book
"Bayesian Reasoning and Machine Learning" by D. Barber
Cambridge University Press (2012)
http://www.cs.ucl.ac.uk/staff/d.barber/brml

Material derived from figures for the book
"Python Data Science Handbook" by J. VanderPlas
O'Reilly Media (2017)
http://shop.oreilly.com/product/0636920034919.do

Material derived from slides for the course
"Machine Learning" by A. Srinivasan

BITS Pilani, Goa, India (2016)

Aims

This lecture will develop your understanding of ensemble methods in machine learning, based on analyses and algorithms covered previously. Following it you should be able to:

- describe the framework of the bias-variance decomposition and some of its practical implications
- describe how ensembles might be used to address the bias and variance components of error
- outline the concept of the stability of a learning algorithm
- describe the ensemble methods of bagging, random forests and boosting
- compare the operation of these methods in terms of the bias and variance components of error

Introduction

In previous lectures, introduced some theoretical ideas about limits on machine learning. But do these have any practical impact ?

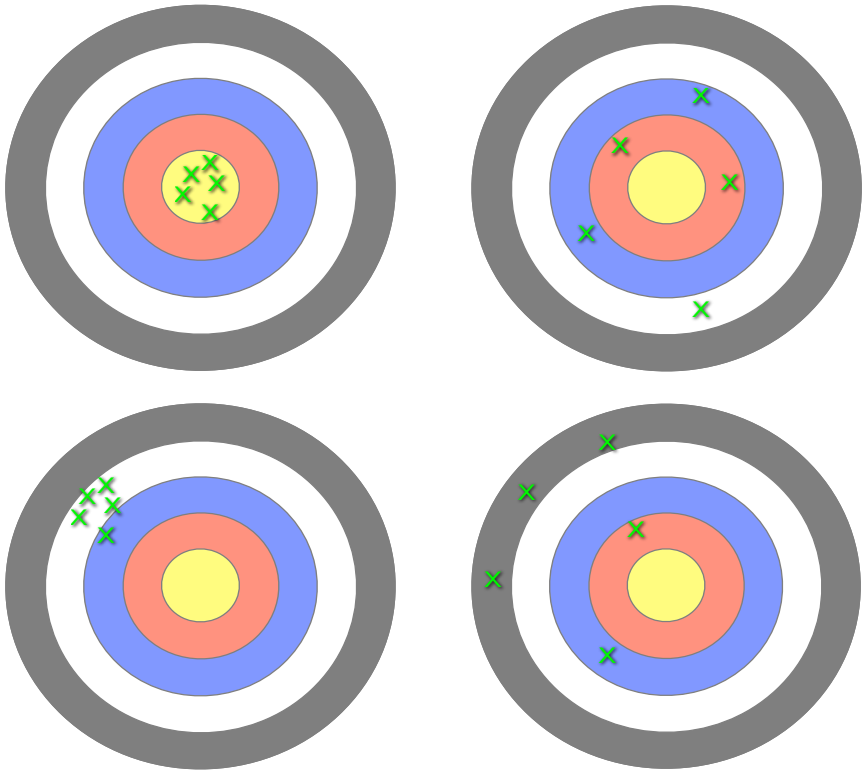
The answer is **yes** !

- The bias-variance decomposition of error can be a tool for thinking about how to reduce error in learning
- Take a learning algorithm and ask:
  - how can we reduce its bias ?
  - how can we reduce its variance ?
- Ensemble learning methods can be viewed in this light
- A form of *multi-level* learning: learning a number of base-level models from the data, and learning to combine these models as an ensemble

Review: bias-variance decomposition

- Theoretical tool for analyzing how much specific training set affects performance of classifier
- Assume we have an infinite number of classifiers built from different training sets all of the same size
  - The *bias* of a learning scheme is the expected error due to the mismatch between the learner's hypothesis space and the space of target concepts
  - The *variance* of a learning scheme is the expected error due to the particular training sets used
  - Total expected error $\approx$ bias + variance

Bias and variance



A dartboard metaphor illustrating the concepts of bias and variance. Each dartboard corresponds to a different learning algorithm, and each dart signifies a different training sample. The top row learning algorithms exhibit low bias, on average staying close to the bull's eye (the true function value for a particular x), while the ones on the bottom row have high bias. The left column shows low variance and the right column high variance.

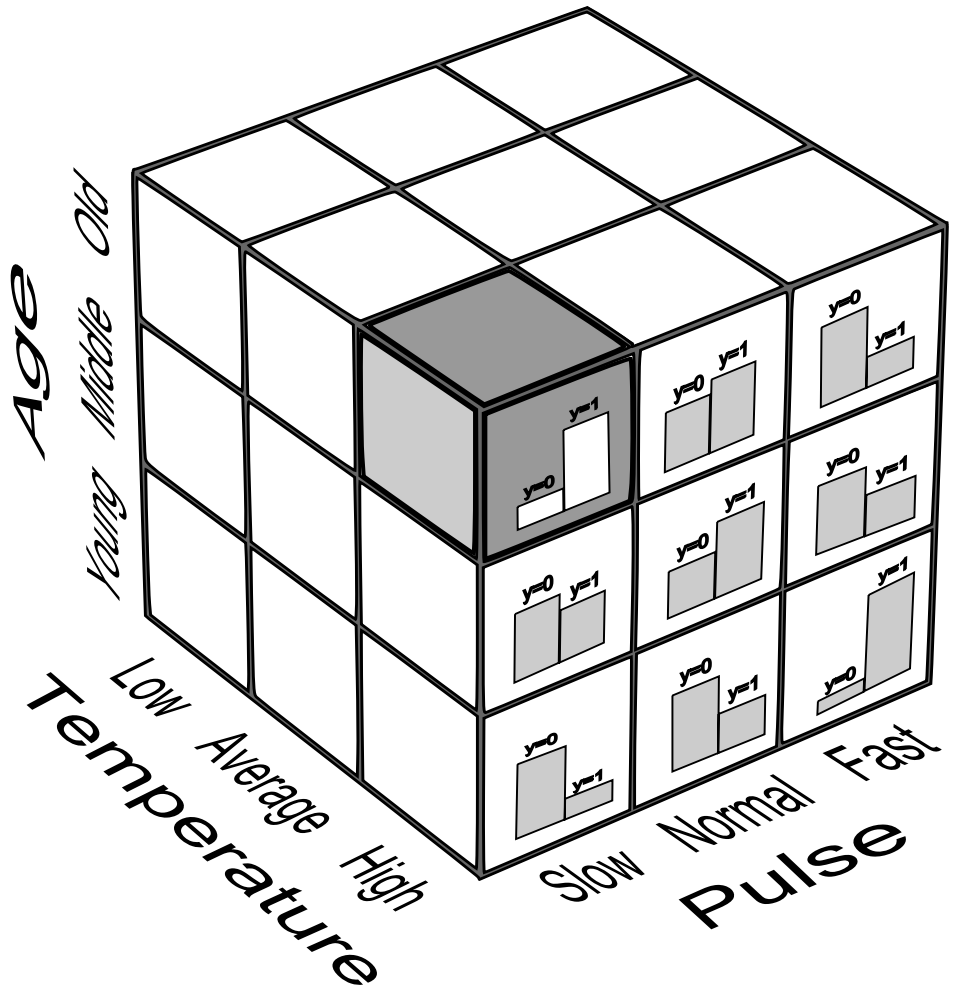
Bias-variance with "Big Data"

The following 4 slides are due to Prof. G. Webb, Monash U.

- high bias algorithms often used for efficiency
  - why ?
- big data can reduce variance
  - why ?

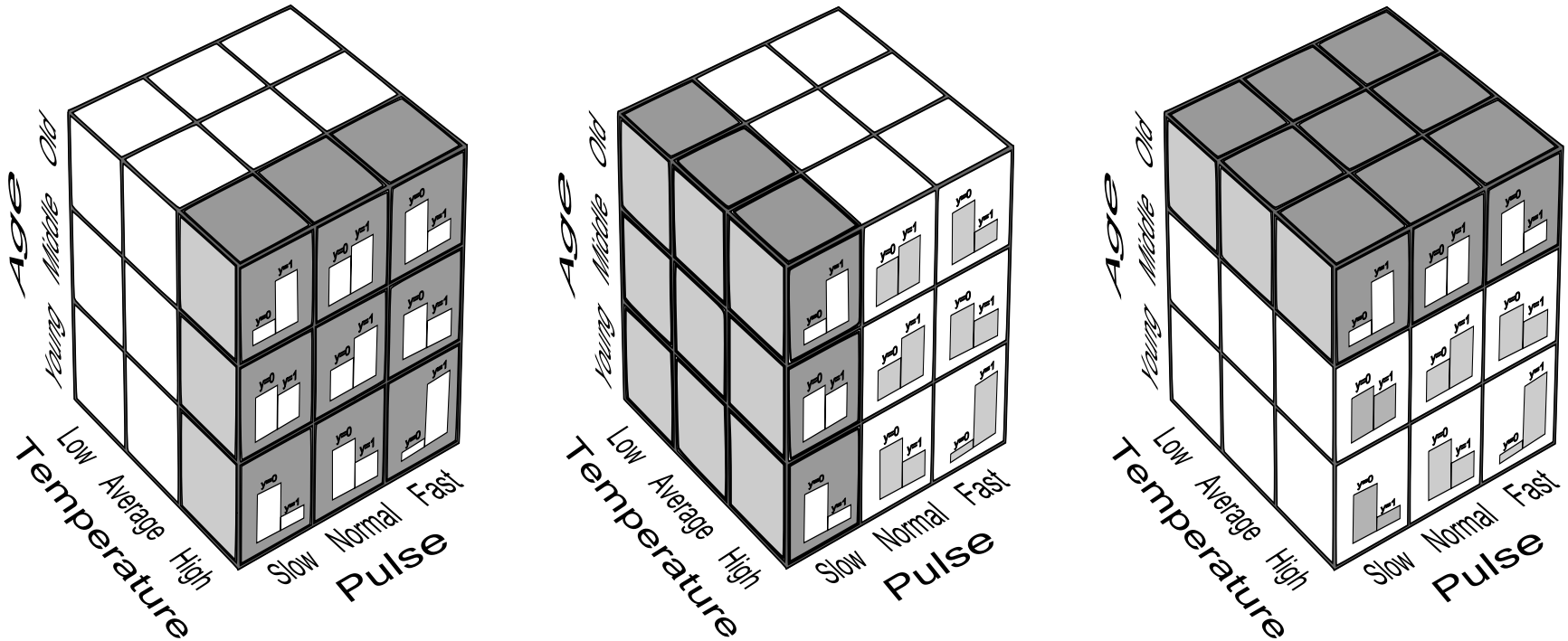
Bias-variance with "Big Data"

Suppose we have a low bias representation (e.g., all conjunctive concepts), but such concepts may not always occur frequently in small datasets:



Bias-variance with "Big Data"

So we can increase bias – e.g., by Naive Bayes-type conditional independence assumptions – but this forces averaging of class distributions over all "small concepts":



Bias-variance with "Big Data"

"Big Data" may help to resolve the bias-variance dilemma:

- high bias algorithms are often used for efficiency
  - usually simpler to compute
- big data can reduce variance
  - "small" concepts will occur more frequently
  - low bias algorithms can find them in each sample
  - but: how to compute efficiently ?

This is still largely an open problem!

Stability

- for a given data distribution $\mathcal{D}$
- train algorithm L on training sets S_1, S_2 sampled from $\mathcal{D}$
- expect that the model from L should be the same (or very similar) on both S_1 and S_2
- if so, we say that L is a *stable* learning algorithm
- otherwise it is unstable
- typical stable algorithm: kNN (for some k)
- typical unstable algorithm: decision-tree learning

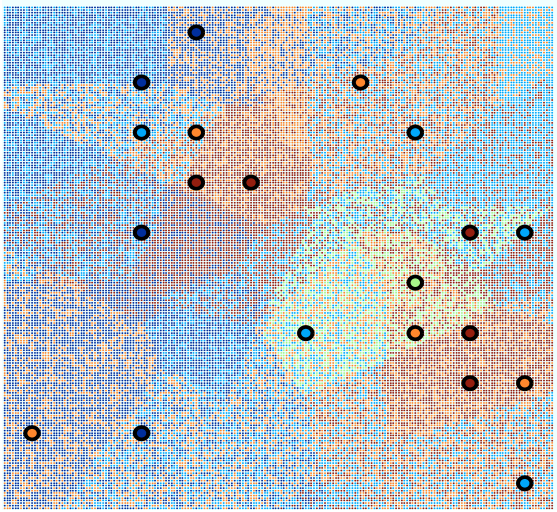
Turney, P. *"Bias and the Quantification of Stability"*

Stability

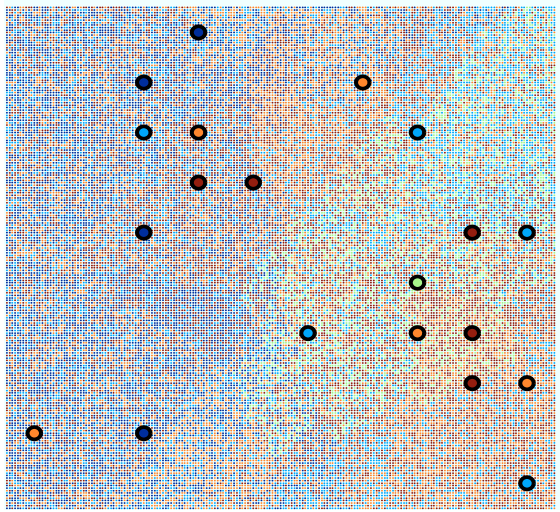
- stable algorithms would typically have high bias
- unstable algorithms would typically have high variance
- but: take care to consider effect of parameters, e.g., in kNN
  - 1NN perfectly separates training data, so low bias but high variance
  - By increasing the number of neighbours k we increase bias and decrease variance (what happens when $k = n$?)
  - Every test instance will have the same number of neighbours, and the class probability vectors will all be the same !

Three-, five- and seven-nearest neighbour

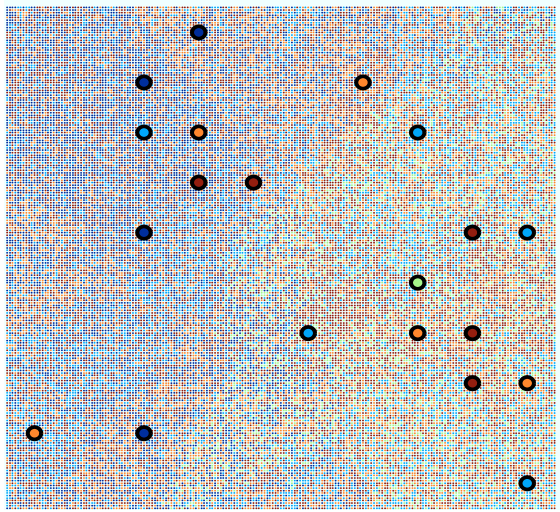
Decision regions of k-nearest neighbour classifiers; the shading represents the predicted probability distribution over the five classes.



3-nearest neighbour



5-nearest neighbour



7-nearest neighbour

Illustrates the effect of varying k on stability (i.e., bias and variance).

Ensemble methods

In essence, ensemble methods in machine learning have the following two things in common:

- they construct multiple, diverse predictive models from adapted versions of the training data (most often reweighted or resampled);
- they combine the predictions of these models in some way, often by simple averaging or voting (possibly weighted).

Ensembles: combining multiple models

- Basic idea of *ensembles* or "multi-level" learning schemes: build different "experts" and let them vote
- Advantage: often improves predictive performance
- Disadvantage: produces output that is very hard to interpret
- Notable schemes: bagging, random forests, boosting
  - can be applied to both classification and numeric prediction problems

Bootstrap error estimation

This is a standard "resampling" technique from statistics. Can be used to estimate a parameter of interest, e.g., error rate of a learning method on a data set.

- sampling from data set *with replacement*
- e.g. sample from n instances, with replacement, n times to generate another data set of n instances
- (almost certainly) new data set contains some duplicate instances
- and does not contain others – used as the test set
- chance of *not* being picked $(1 - \frac{1}{n})^n \approx e^{-1} = 0.368$
- 0.632 training set
- error estimate $= 0.632 \times \text{err}_{test} + 0.368 \times \text{err}_{train}$
- repeat and average with different bootstrap samples

Bagging

"**B**ootstrap **Agg**regation"

- Employs simplest way of combining predictions: voting/averaging
- Each model receives equal weight
- Generalized version of bagging:
  - Sample several training sets of size n (instead of just having one training set of size n)
  - Build a classifier for each training set
  - Combine the classifiers' predictions
- This improves performance in almost all cases if learning scheme is unstable (i.e. decision trees)

Bagging

- Bagging reduces variance by voting/averaging, thus reducing the overall expected error
  - In the case of classification there are pathological situations where the overall error might increase
  - Usually, the more classifiers the better
- Problem: we only have one dataset!
- Solution: generate new datasets of size n by sampling with replacement from original dataset
- Can help a lot if data is noisy

Bagging in a nutshell

Learning (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 "ensemble" class.

What is the ensemble class ? The class that has been predicted most often (for classification, i.e., the majority vote or mode), or the mean of the output class values (for regression).

Bagging more precisely

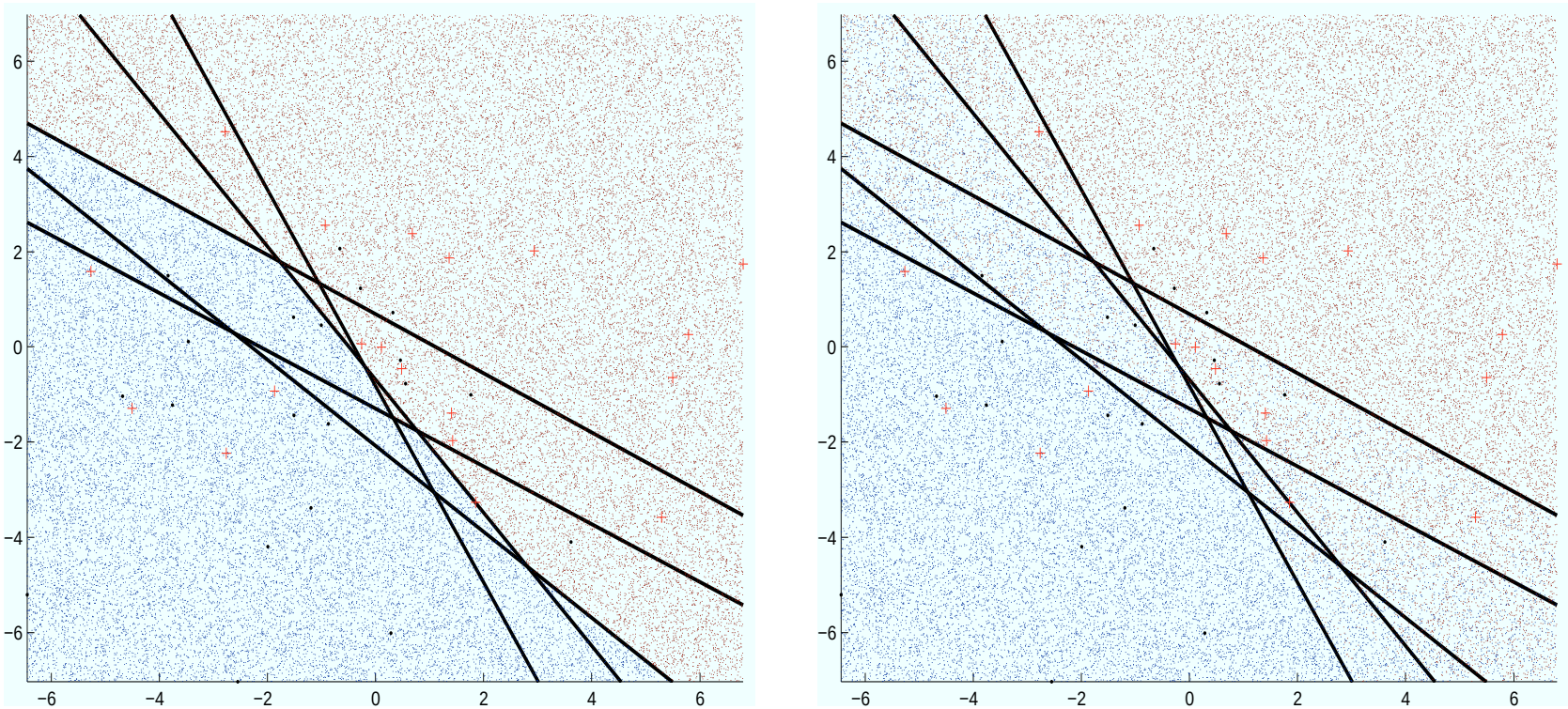
Algorithm Bagging$(D, T, \mathcal{A})$ // train ensemble from bootstrap samples

Input: dataset D; ensemble size T; learning algorithm $\mathcal{A}$.

Output: set of models; predictions to be combined by voting or averaging.

1 **for** $t = 1$ to T **do**
2 | bootstrap sample D_t from D by sampling $|D|$ examples with replacement
3 | run $\mathcal{A}$ on D_t to produce a model M_t
4 **end**
5 **return** $\{M_t | 1 \leq t \leq T\}$

Bagging linear classifiers



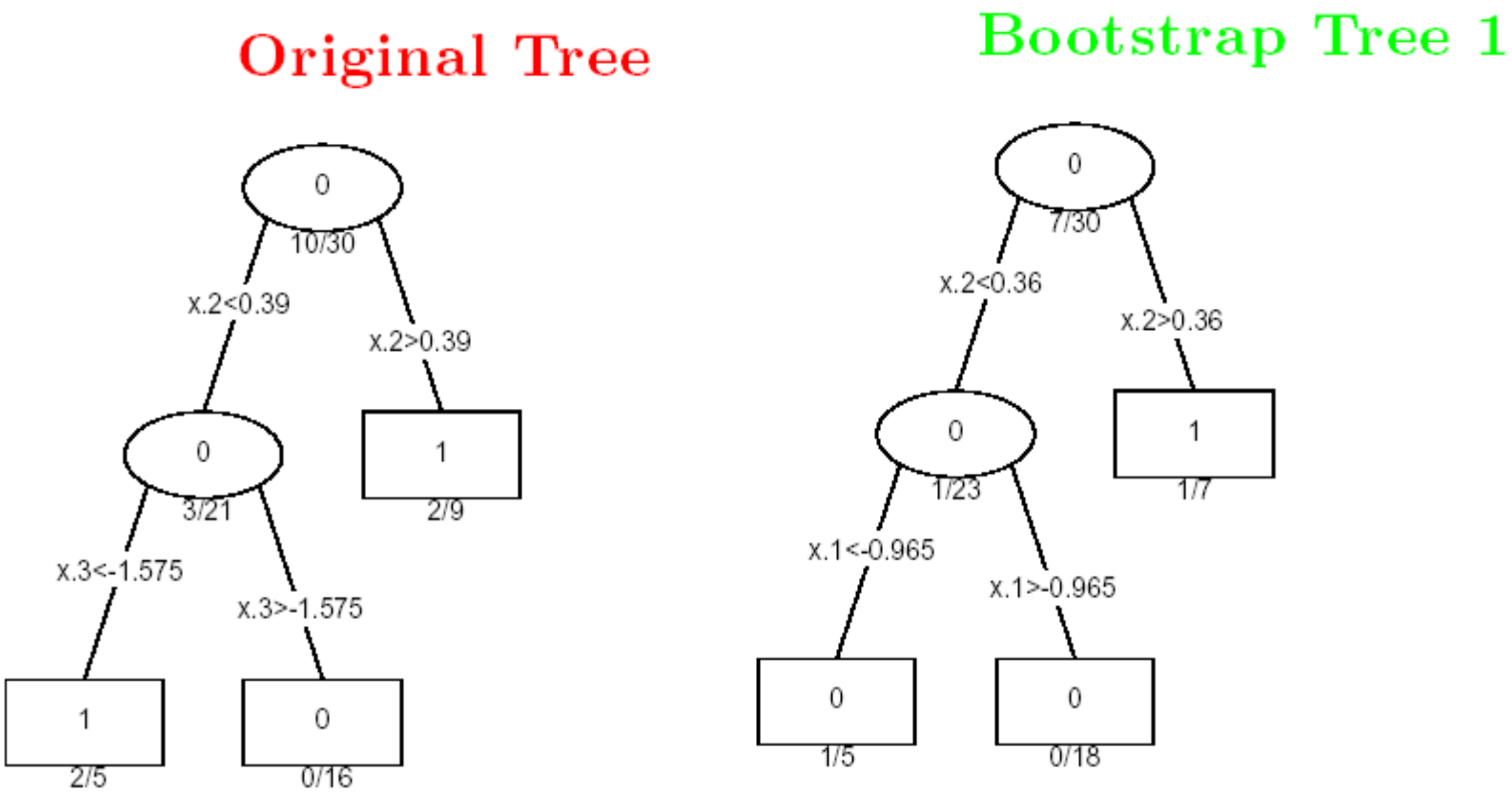
(left) An ensemble of five *basic linear classifiers* built from bootstrap samples with bagging. The decision rule is majority vote, leading to a piecewise linear decision boundary. (right) If we turn the votes into probabilities, we see the ensemble is effectively grouping instances in different ways, with each segment obtaining a slightly different probability.

Bagging trees

An experiment with simulated data:

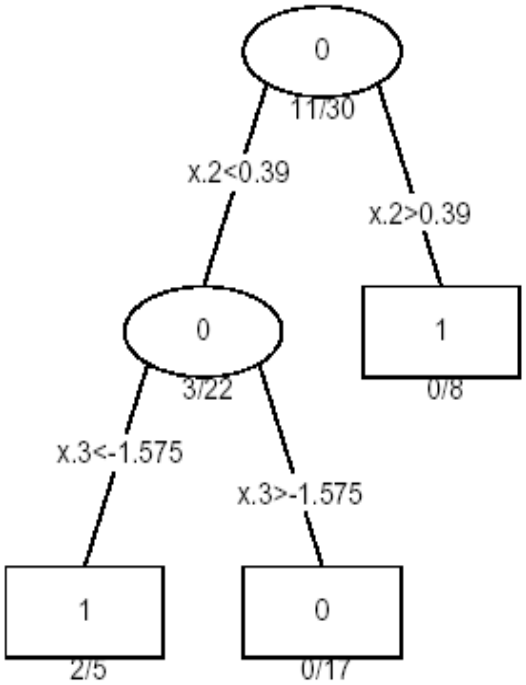
- sample of size $n = 30$, two classes, five features
- $Pr(Y = 1|x_1 \leq 0.5) = 0.2$ and $Pr(Y = 1|x_1 > 0.5) = 0.8)$
- test sample of size 2000 from same population
- fit classification trees to training sample, 200 bootstrap samples
- trees are different (tree induction is *unstable*)
- therefore have high variance
- averaging reduces variance and leaves bias unchanged
- (graph: test error for original and bagged trees, with green – vote; purple – average probabilities)

Bagging trees

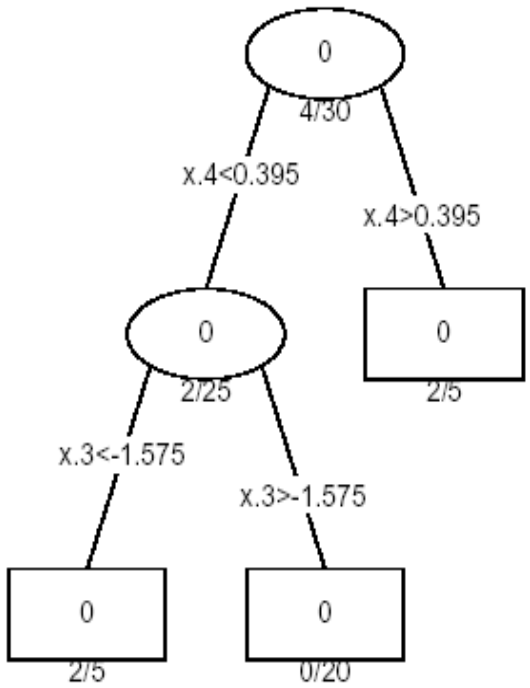


Bagging trees

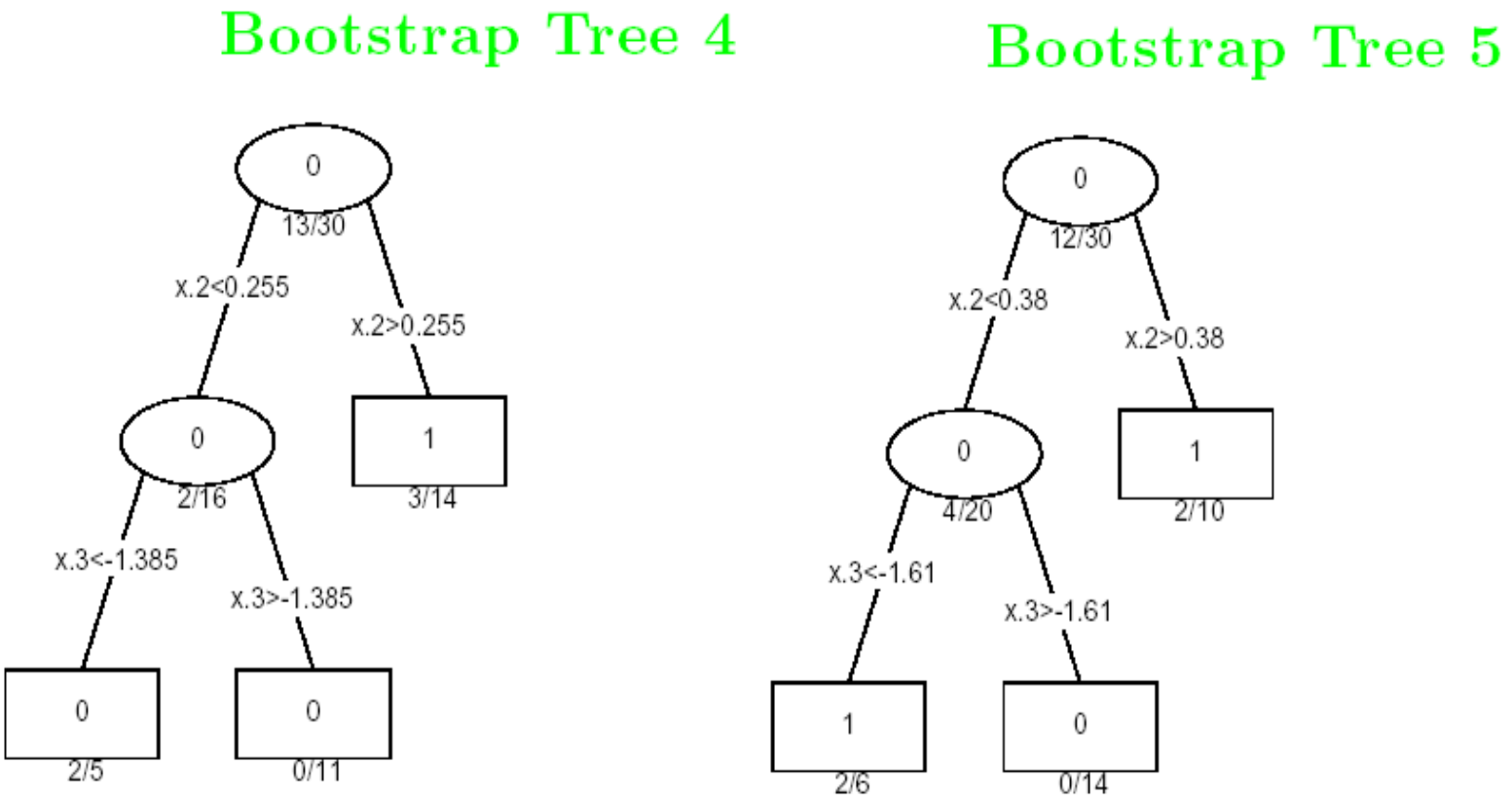
Bootstrap Tree 2



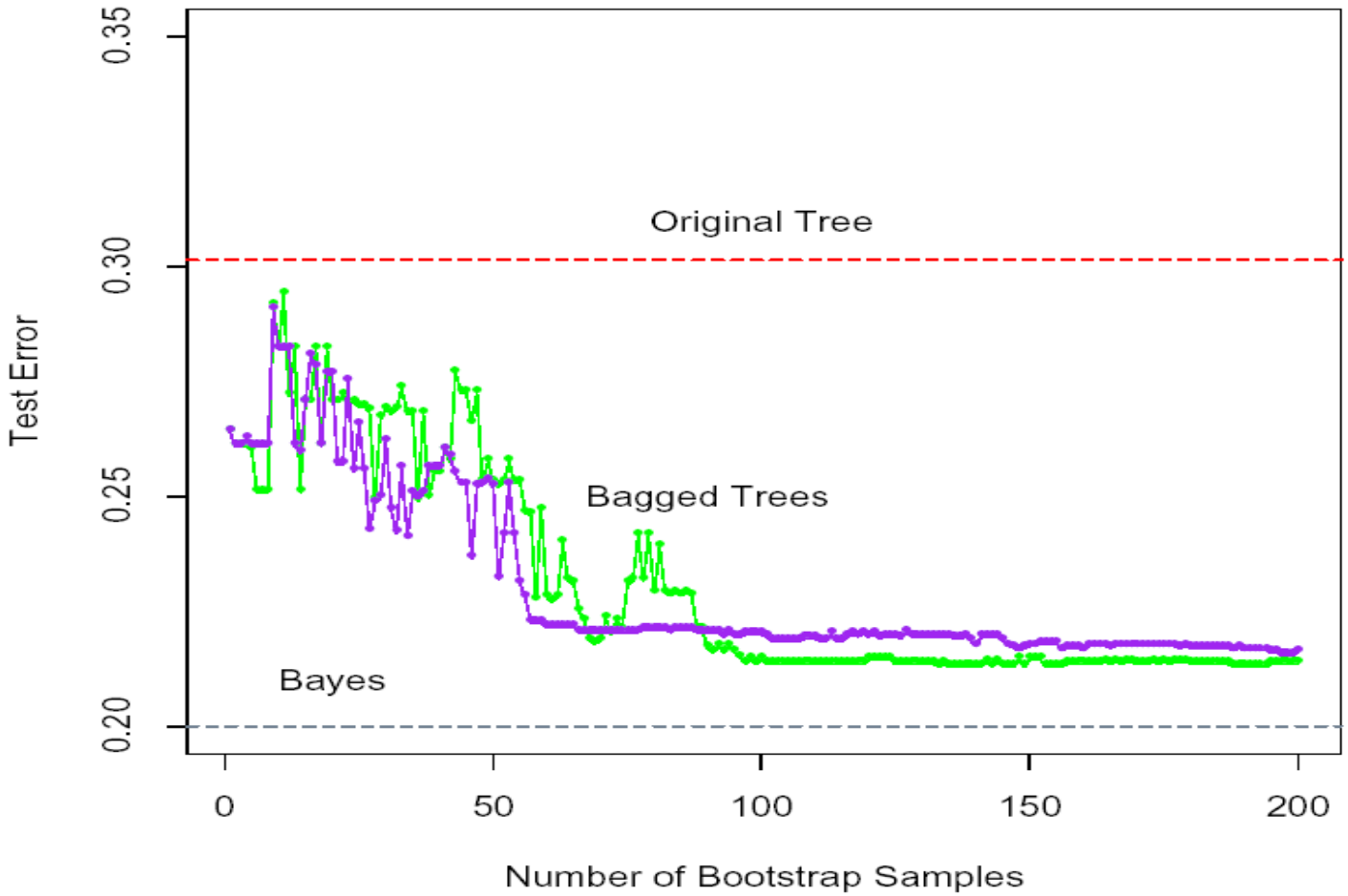
Bootstrap Tree 3



Bagging trees



Bagging trees



Bagging trees

The news is not all good:

- when we bag a model, any simple structure is lost
- this is because a bagged tree is no longer a tree ...
- ... but a forest
- although bagged trees can be mapped back to a single tree ...
- ... this reduces claim to comprehensibility

- *stable* models like nearest neighbour not very affected by bagging
- *unstable* models like trees most affected by bagging
- usually, their design for interpretability (bias) leads to instability
- more recently, *random forests* (see Breiman's web-site)

Random Forests

Algorithm RandomForest(D, T, d) // train ensemble of randomized trees

Input: data set D; ensemble size T; subspace dimension d.

Output: set of models; predictions to be combined by voting or averaging.

for $t = 1$ to T **do**
    bootstrap sample D_t from D by sampling $|D|$ examples with replacement
    select d features at random and reduce dimensionality of D_t accordingly
    train a tree model M_t on D_t without pruning
end
return $\{M_t | 1 \leq t \leq T\}$

Random Forests

Leo Breiman's Random Forests algorithm is essentially like Bagging for trees, except the ensemble of tree models is trained from bootstrap samples and random subspaces.

- each tree in the forest is learned from
  - a bootstrap sample, i.e., sample from the training set with replacement
  - a subspace sample, i.e., randomly sample a subset of features
- advantage: forces more diversity among trees in ensemble
- advantage: less time to train since only consider a subset of features

Note: combining linear classifiers in an ensemble gives a piecewise linear (i.e., non-linear) model, whereas multiple trees can be combined into a single tree.

Boosting

- Also uses voting/averaging but each model is *weighted* according to their performance
- Iterative procedure: new models are influenced by performance of previously built ones
  - New model is encouraged to become "expert" for instances classified incorrectly by earlier models
  - Intuitive justification: models should be experts that complement each other
- There are several variants of this algorithm ...

The strength of weak learnability

- Learner produces a binary $[-1, +1]$ classifier h with error rate $\epsilon < 0.5$.
- In some sense h is "useful", i.e., better than random !
- **strong** learner if $\epsilon < 0.5$ and ϵ "close" to zero.
- **weak** learner if $\epsilon < 0.5$ and ϵ "close" to 0.5.
- Question (arising from Valiant's PAC framework):
  is there a procedure to convert a weak learner into a strong learner ?

The strength of weak learnability

Schapire (1990) - first boosting algorithm.

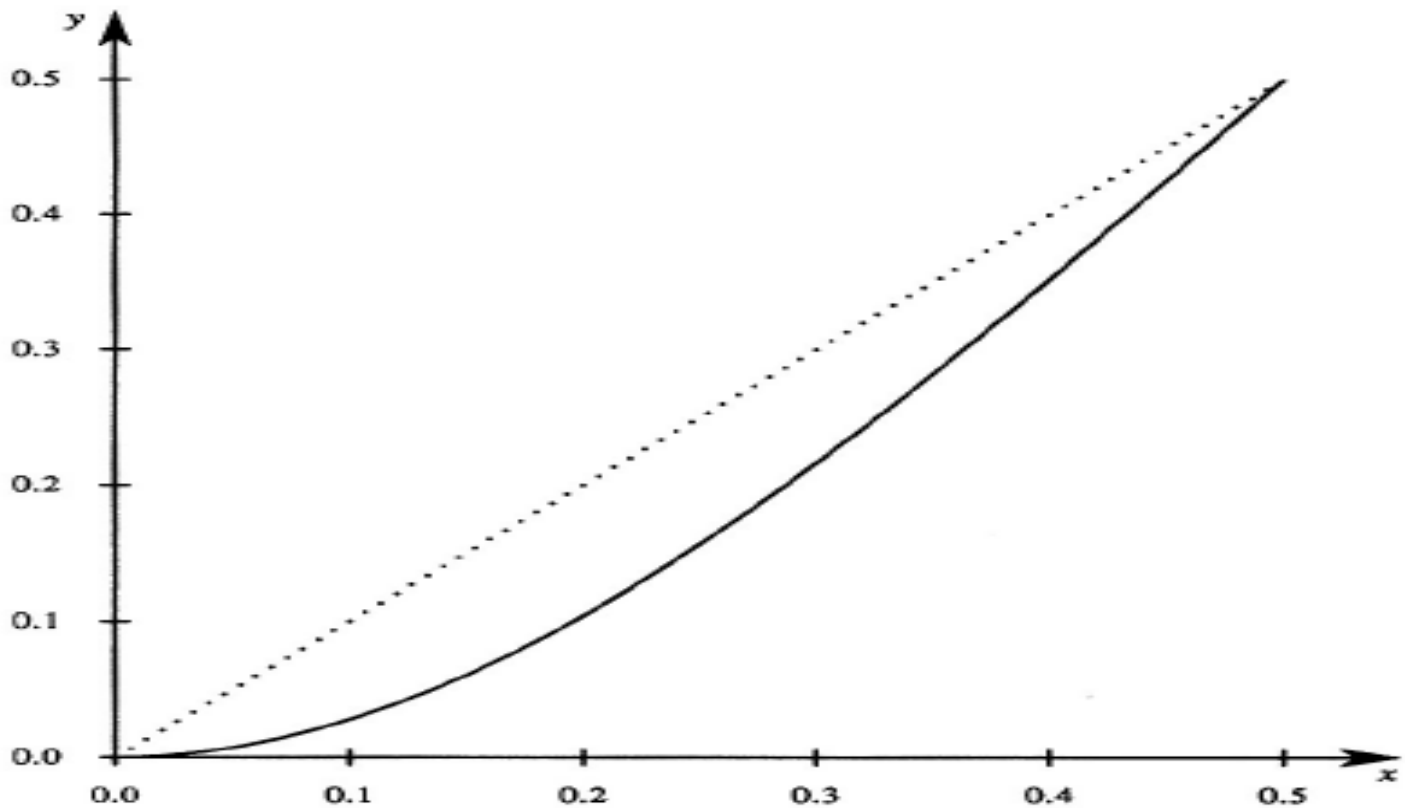
Method:

- weak learner learns initial hypothesis h_1 from N examples
- next learns hypothesis h_2 from new set of N examples, half of which are misclassified by h_1
- then learns hypothesis h_3 from N examples for which h_1 and h_2 *disagree*
- "boosted" hypothesis h gives voted prediction on instance x:
  - if $h_1(x) = h_2(x)$ then return agreed prediction, else
  - return $h_3(x)$

Result: if h_1 has error rate $\epsilon < 0.5$ then error of h bounded by $3\epsilon^2 - 2\epsilon^3$, i.e., better than ϵ (see next slide).

Schapire showed that weak learners *can* be boosted into strong learners.

Boosting a weak learner reduces error



A graph of the function $g(x) = 3x^2 - 2x^3$.

A general boosting method

- original version: after initial hypothesis, each subsequent hypothesis has to "focus" on errors made by previous hypotheses
- general version: extend from 3 hypotheses to many
- how to focus current hypothesis on errors of previous hypotheses ?
- apply *weights* to misclassified *examples*
- called *adaptive boosting*

Weight updates in boosting

- Suppose a *linear classifier* achieves performance as in the first contingency table. The error rate is $\epsilon = (9 + 16)/100 = 0.25$.
- We want to give half the weight to the misclassified examples. The following weight updates achieve this: a factor $1/2\epsilon = 2$ for for the misclassified examples and $1/2(1 - \epsilon) = 2/3$ for the correctly classified examples.

	Predicted $\oplus$	*Predicted* $\ominus$	
Actual $\oplus$	**24**	**16**	40
Actual $\ominus$	**9**	**51**	60
	33	67	100

Weight updates in boosting

- Taking these updated weights into account leads to the contingency table below, which has a (weighted) error rate of 0.5.

	$\oplus$	$\ominus$	
$\oplus$	**16**	**32**	48
$\ominus$	**18**	**34**	52
	34	66	100

Boosting

Algorithm Boosting$(D, T, \mathcal{A})$ // train binary classifier ensemble, reweighting d

Input: data set D; ensemble size T; learning algorithm $\mathcal{A}$

Output: weighted ensemble of models

$w_{1i} \leftarrow 1/|D|$ for all $x_i \in D$

for $t = 1$ to T **do**

  run $\mathcal{A}$ on D with weights w_{ti} to produce a model M_t

  calculate weighted error ϵ_t

  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$

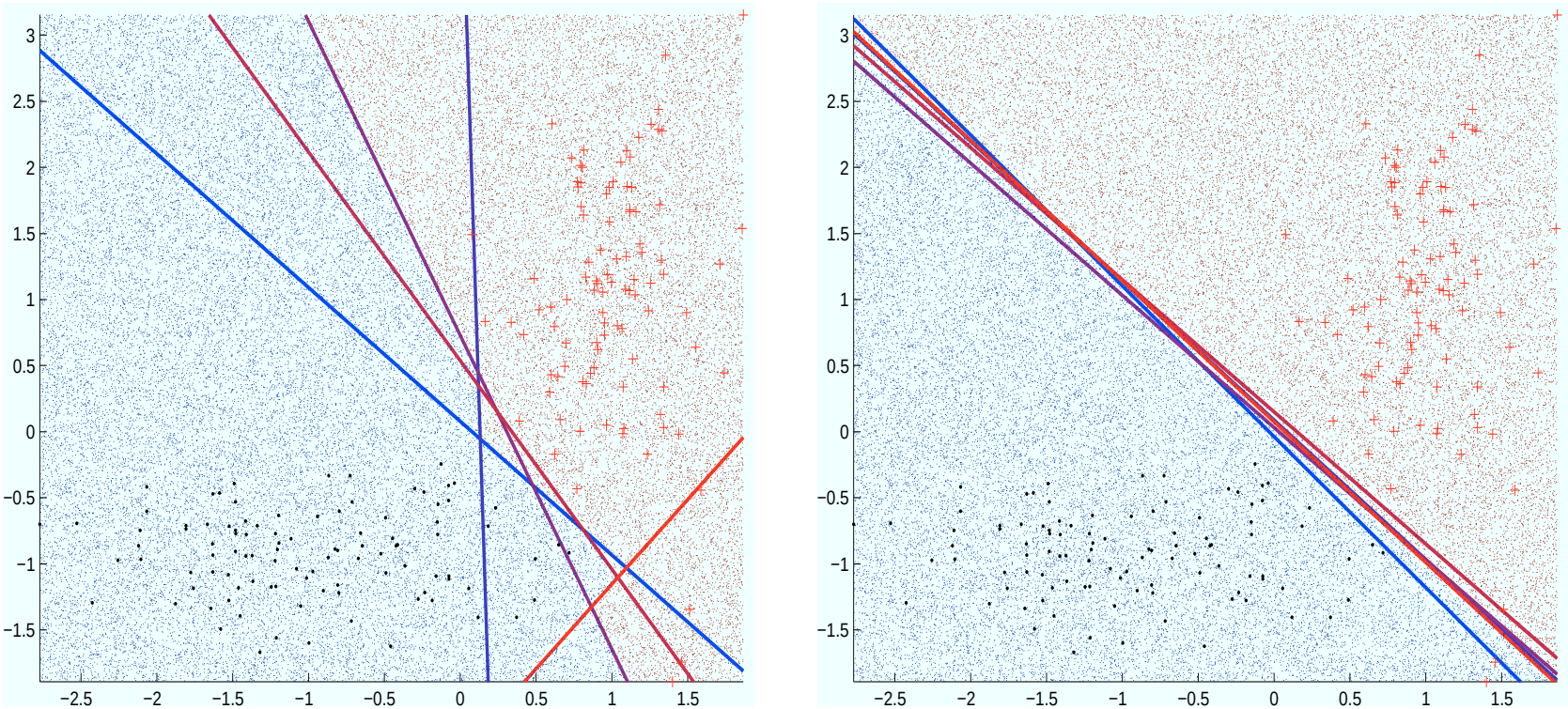
  $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$

  $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$

end

return $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$

Boosting



(left) An ensemble of five boosted *basic linear classifiers* with majority vote. The linear classifiers were learned from blue to red; none of them achieves zero training error, but the ensemble does. (right) Applying bagging results in a much more homogeneous ensemble, indicating that there is little diversity in the bootstrap samples.

Why those α_t?

The two weight updates for the misclassified instances and the correctly classified instances can be written as reciprocal terms δ_t and $1/\delta_t$ normalised by some term Z_t:

$$\frac{1}{2\epsilon_t} = \frac{\delta_t}{Z_t} \qquad \frac{1}{2(1-\epsilon_t)} = \frac{1/\delta_t}{Z_t}$$

From this we can derive

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \qquad \delta_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \exp(\alpha_t)$$

So the weight update for misclassified instances is $\exp(\alpha_t)/Z_t$ and for correctly classified instances $\exp(-\alpha_t)/Z_t$. Using the fact that $y_i M_t(x_i) = +1$ for instances correctly classified by model M_t and -1 otherwise, we can write the weight update as

$$w_{(t+1)i} = w_{ti}\frac{\exp(-\alpha_t y_i M_t(x_i))}{Z_t}$$

which is the expression commonly found in the literature.

More on boosting

- 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
- Stems from computational learning theory
- Theoretical result: training error decreases exponentially
- Also: works if base classifiers not too complex and their error doesn't become too large too quickly

A bit more on boosting

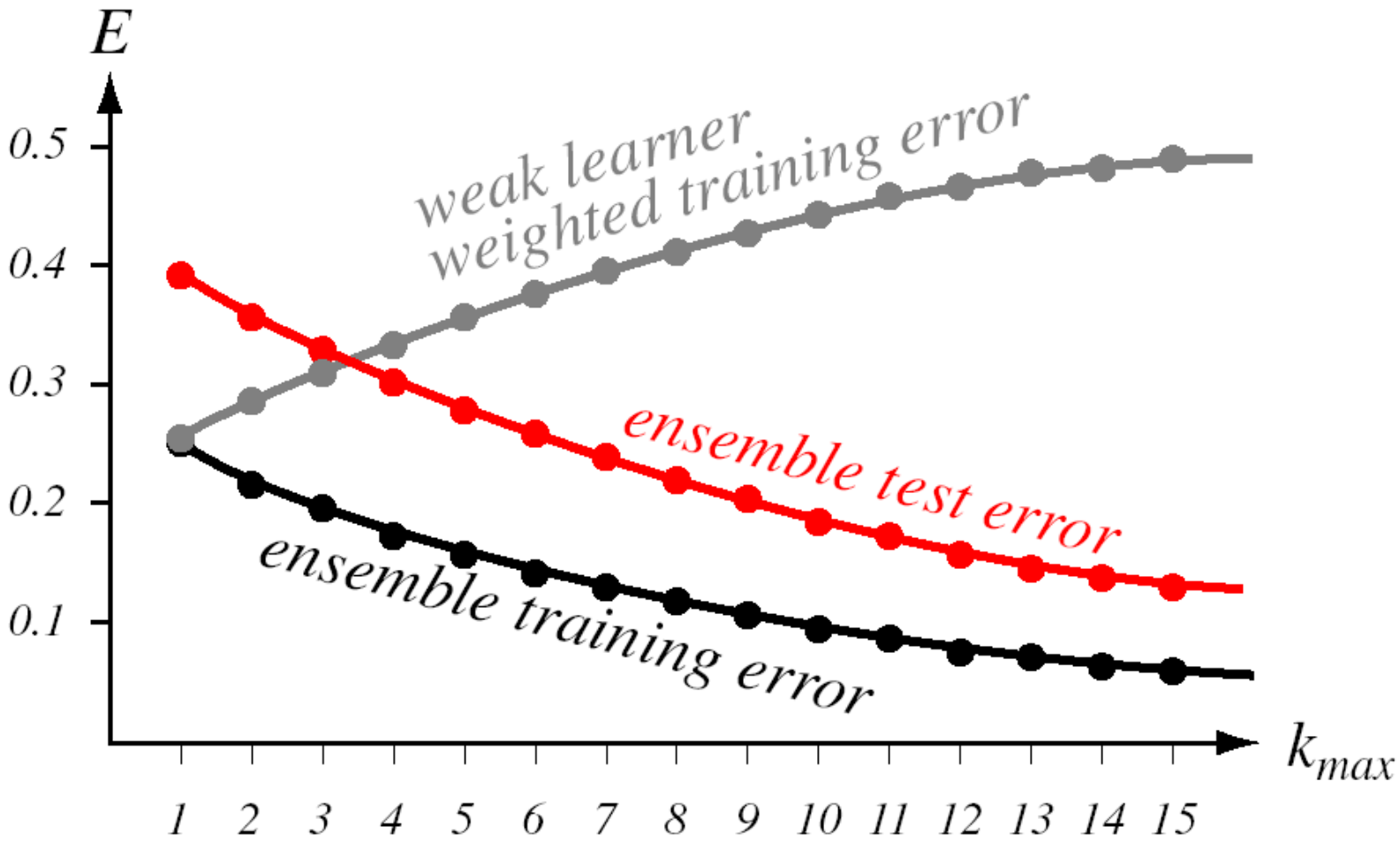
- Puzzling fact: generalization error can decrease long after training error has reached zero
  - Seems to contradict Occam's Razor !
  - However, problem disappears if *margin* (confidence) is considered instead of error
    - Margin: difference between estimated probability for true class and most likely other class (between -1, 1)
- Boosting works with *weak* learners: only condition is that error ϵ doesn't exceed 0.5 (slightly better than random guessing)
- LogitBoost: more sophisticated boosting scheme in Weka (based on additive logistic regression)

Boosting reduces error

Adaboost applied to a weak learning system can reduce the training error exponentially as the number of component classifiers is increased.

- focuses on "difficult" patterns
- training error of successive classifier on its own weighted training set is generally larger than predecessor
- training error of ensemble will decrease
- typically, test error of ensemble will decrease also

Boosting reduces error

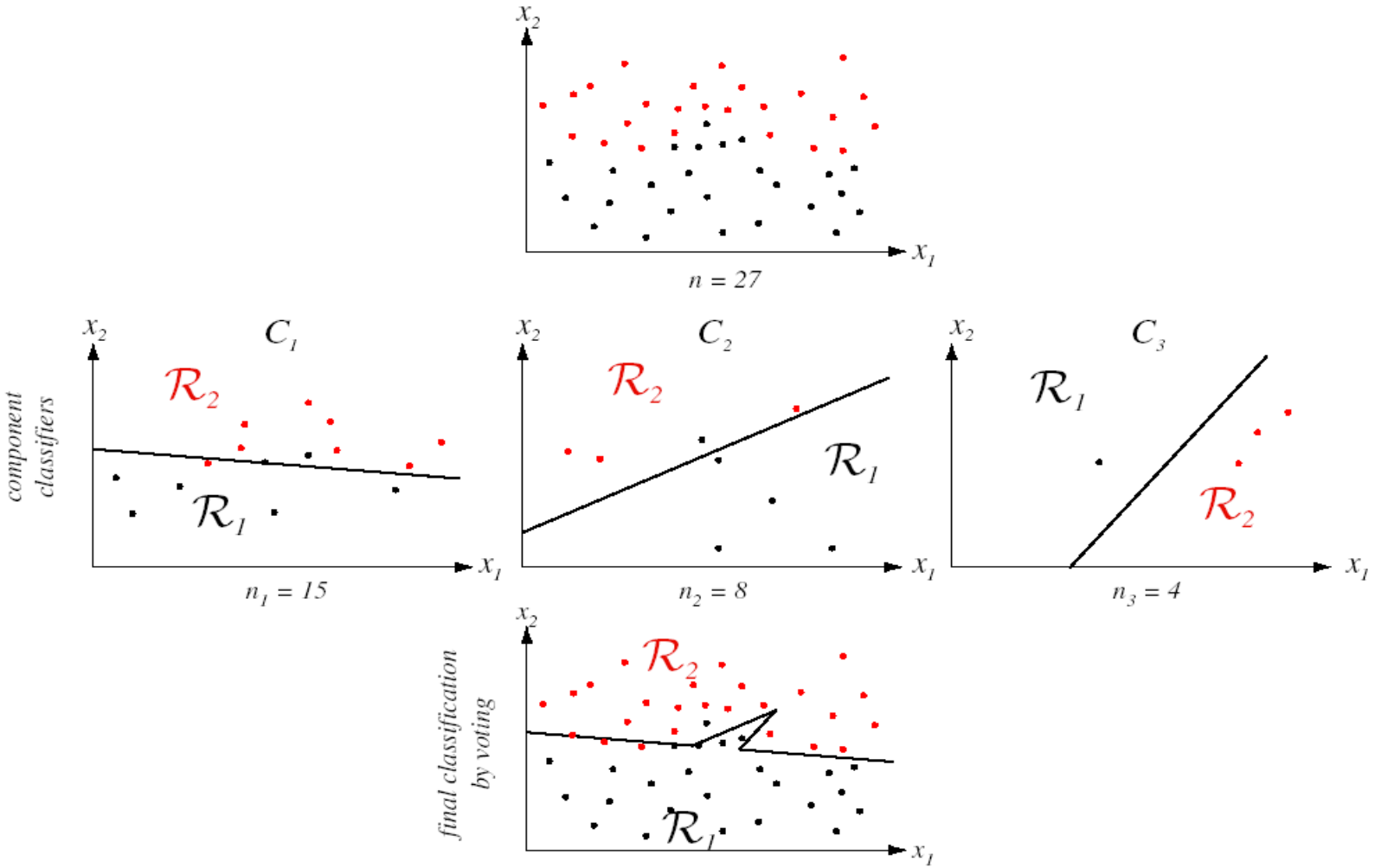


Boosting enlarges the model class

A two-dimensional two-category classification task

- three component linear classifiers
- final classification is by voting component classifiers
- gives a non-linear decision boundary
- each component is a weak learner (slightly better than 0.5)
- ensemble classifier has error lower than any single component
- ensemble classifier has error lower than single classifier on complete training set

Boosting enlarges the model class



Ensemble Learning

Important points to remember

Low-bias models tend to have high variance, and vice versa.

Bagging is predominantly a variance-reduction technique, while boosting is primarily a bias-reduction technique.

This explains why bagging is often used in combination with high-variance models such as tree models (as in Random Forests, whereas boosting is typically used with high-bias models such as linear classifiers or univariate decision trees (also called *decision stumps*.

Ensemble Learning

- Bias-variance decomposition breaks down error, suggests possible fixes to improve learning algorithms
- Stability idea captures aspects of both bias and variance
- Bagging is a simple way to run ensemble methods
- Random Forests are a popular bagging approach for trees
- Boosting has a more theoretically justified basis and may work better in practice to reduce error, but can be susceptible to very noisy data
- Remember: No Free Lunch and other theorems $\rightarrow$ no "magic bullet" for machine learning!