

Ensemble Learning

16s1: COMP9417 Machine Learning and Data Mining

School of Computer Science and Engineering, University of New South Wales

May 17, 2016

Acknowledgements

Material derived from slides for the book

“Machine Learning” by T. Mitchell, McGraw-Hill (1997)

<http://www-2.cs.cmu.edu/~tom/mlbook.html>

and slides by Andrew W. Moore available at

<http://www.cs.cmu.edu/~awm/tutorials>

and the book Data Mining, Ian H. Witten and Eibe Frank,

Morgan Kauffman, 2000. <http://www.cs.waikato.ac.nz/ml/weka>

and the book Pattern Classification, Richard O. Duda, Peter E. Hart,

and David G. Stork. Copyright (c) 2001 by John Wiley & Sons, Inc.

and the book “Elements of Statistical Learning”, Trevor Hastie,

Robert Tibshirani and Jerome Friedman. (c) 2001, Springer.

and from slides for the book

“Machine Learning” by P. Flach

Cambridge University Press (2012)

<http://cs.bris.ac.uk/~flach/mlbook>

Aims

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

- ☞ 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
- ☞ contrast the operation of these methods in terms of the bias and variance components of error

In Weka, see: Bagging, RandomForest, AdaBoost.M1

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
- ☞ They can be seen as a form of *multi-level* learning, where base-level models are learned from the data, and then their combination as an ensemble is learned

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

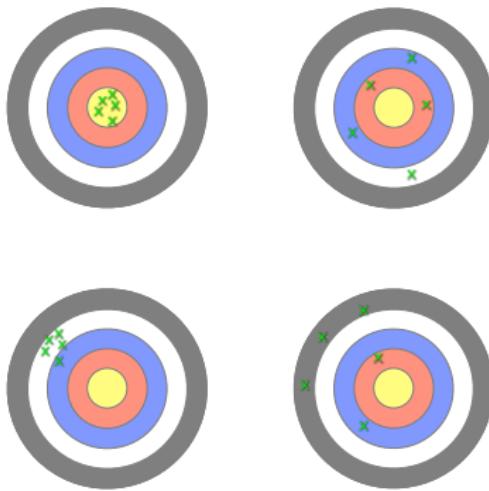
The *variance* of a learning scheme is the expected error due to the particular training sets used

Total expected error \approx bias + variance



Figure 3.3, p.94

★ Bias and variance



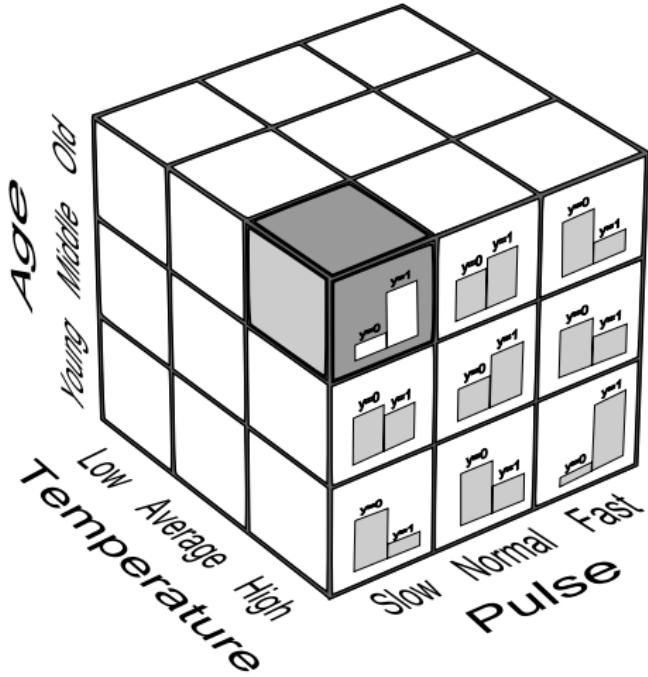
A dartboard metaphor illustrating the concepts of bias and variance. Each dartboard corresponds to a different learning algorithm, and each dart signifies a different training sample. The top row learning algorithms exhibit low bias, staying close to the bull's eye (the true function value for a particular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

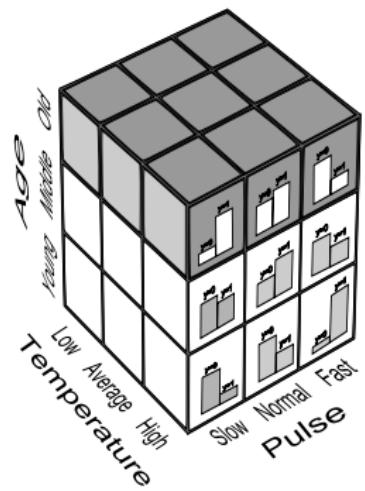
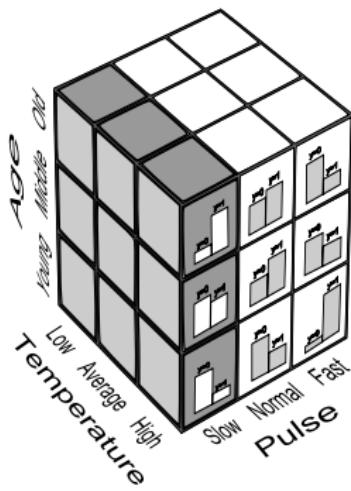
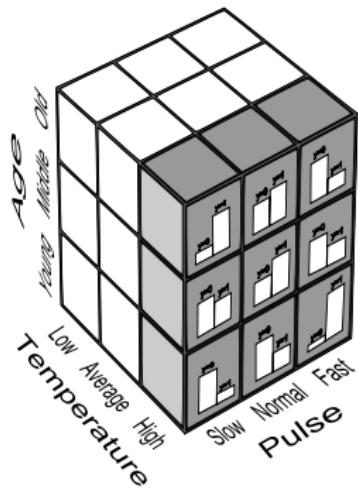
The following 4 slides are due to Prof. G. Webb, Monash U.

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

Bias-variance with “Big Data”



Bias-variance with “Big Data”



- ☞ high bias algorithms 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 ?

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: k NN (for some k)
- ☞ typical unstable algorithm: decision-tree learning

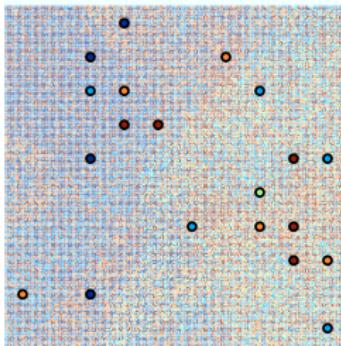
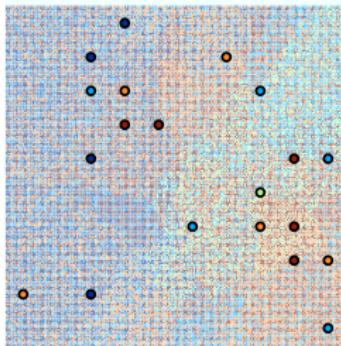
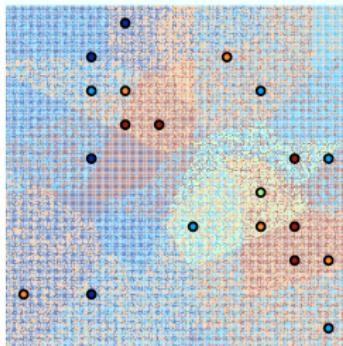
Turney, P. "Bias and the Quantification of Stability"

- ☞ 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 k NN
 - ☞ 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 !



Figure 8.9, p.244

Three-, five- and seven-nearest neighbour



(left) Decision regions of a 3-nearest neighbour classifier; the shading represents the predicted probability distribution over the five classes. **(middle)** 5-nearest neighbour.

(right) 7-nearest neighbour.

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 “meta” 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

Estimating 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 = $0.632 \times \text{err}_{\text{test}} + 0.368 \times \text{err}_{\text{train}}$

repeat and average with different bootstrap samples

“Bootstrap Aggregation”

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 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 algorithm schema

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 class that has been predicted most often.



Algorithm Bagging(D, T, \mathcal{A}) – train an ensemble of models from bootstrap samples.

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

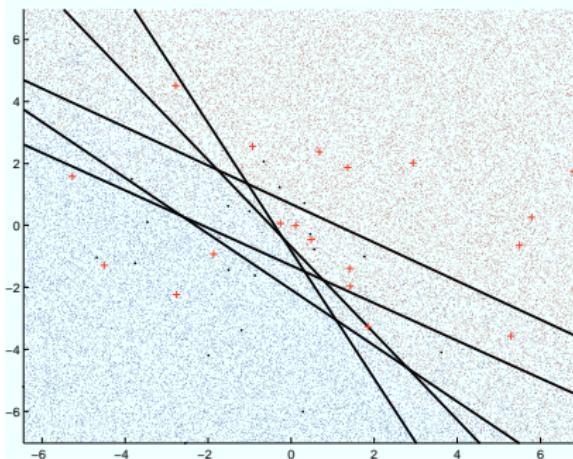
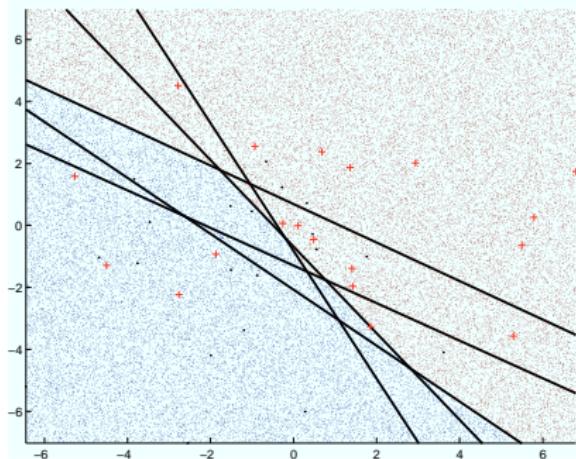
Output : ensemble of models whose predictions are to be combined by voting or averaging.

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



Figure 11.1, p.332

Bagging



(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 a grouping model: each instance space segment obtains 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*)

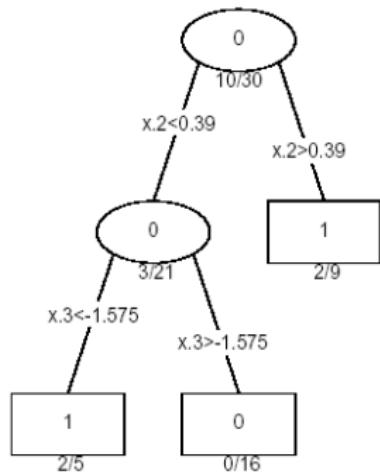
therefore have high variance

averaging reduces variance and leaves bias unchanged

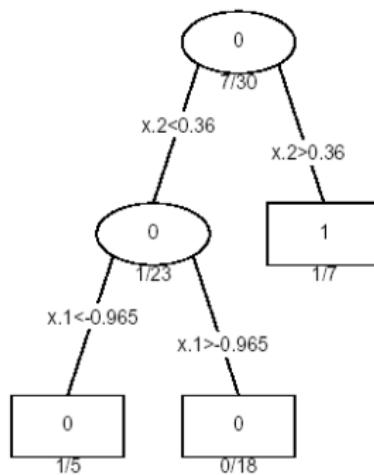
(graph: test error for original and bagged trees, with green – vote; purple – average probabilities)

Bagging trees

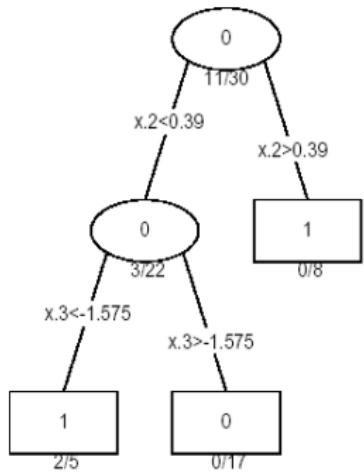
Original Tree



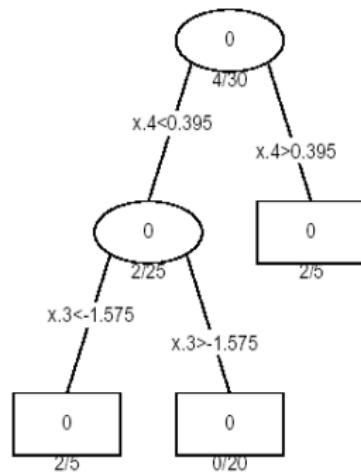
Bootstrap Tree 1



Bootstrap Tree 2

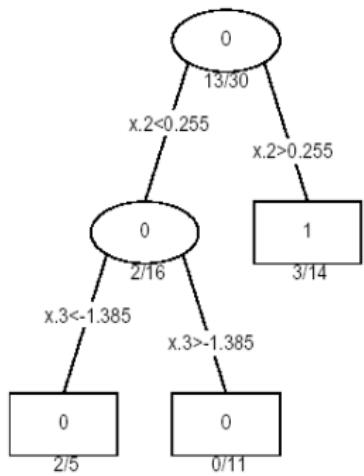


Bootstrap Tree 3

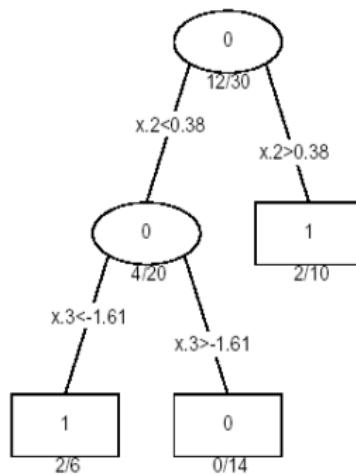


Bagging trees

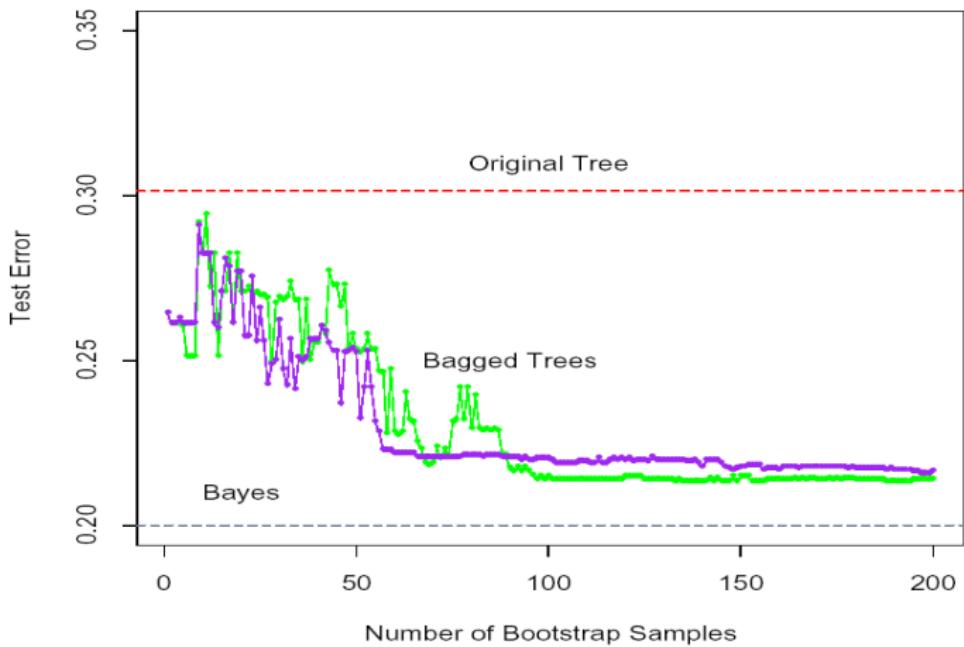
Bootstrap Tree 4



Bootstrap Tree 5



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)



Algorithm $\text{RandomForest}(D, T, d)$ – train an ensemble of tree models from bootstrap samples and random subspaces.

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

Output : ensemble of tree models whose predictions are to be combined by voting or averaging.

```
1 for  $t = 1$  to  $T$  do
2     build a bootstrap sample  $D_t$  from  $D$  by sampling  $|D|$  data points with
    replacement;
3     select  $d$  features at random and reduce dimensionality of  $D_t$  accordingly;
4     train a tree model  $M_t$  on  $D_t$  without pruning;
5 end
6 return  $\{M_t | 1 \leq t \leq T\}$ 
```

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

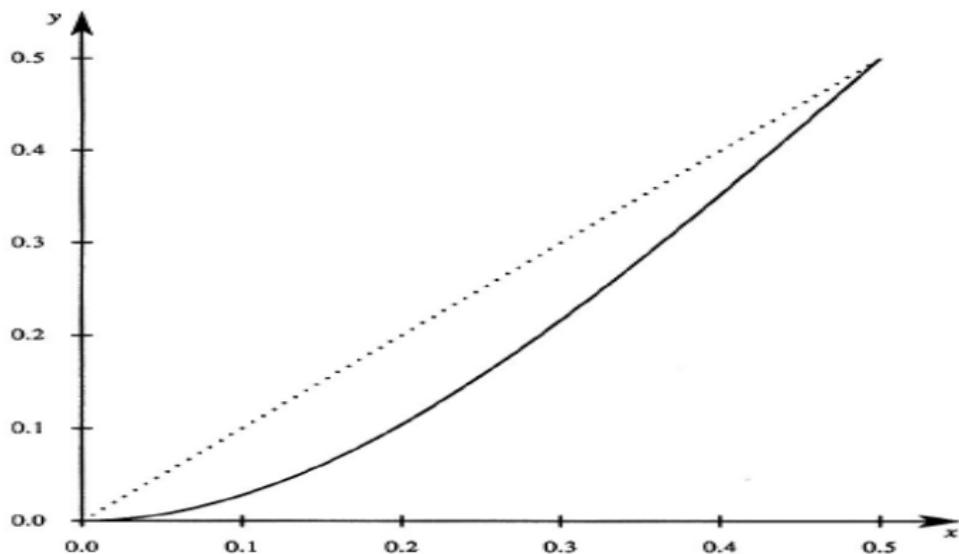
“boosted” hypothesis h gives voted prediction on instance x :

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*



Example 11.1, p.334

Weight updates in boosting

- ☞ Suppose a linear classifier achieves performance as in the contingency table on the left. 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 the misclassified examples and $1/2(1 - \epsilon) = 2/3$ for the correctly classified examples.

	<i>Predicted</i> \oplus	<i>Predicted</i> \ominus		\oplus	\ominus	
<i>Actual</i> \oplus	24	16	40	16	32	48
<i>Actual</i> \ominus	9	51	60	18	34	52
	33	67	100	34	66	100

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



Algorithm 11.3, p.335

Boosting

Algorithm Boosting(D, T, \mathcal{A}) – train an ensemble of binary classifiers from reweighted training sets.

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

Output : weighted ensemble of models.

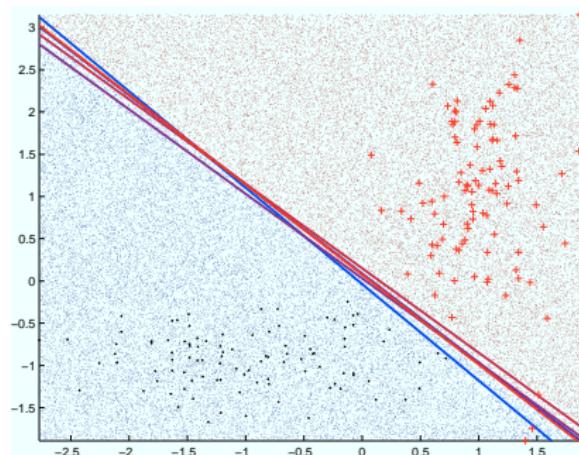
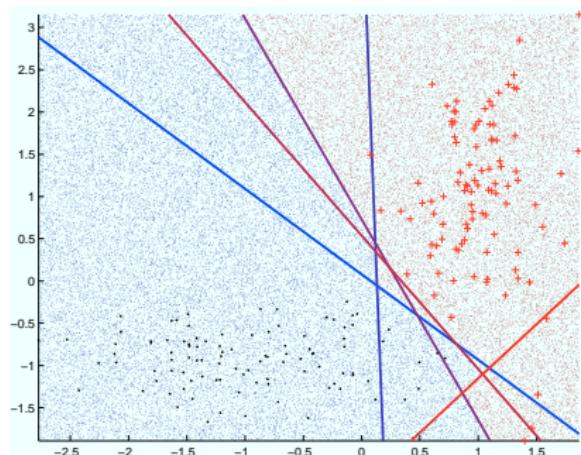
```

1  $w_{1i} \leftarrow 1/|D|$  for all  $x_i \in D$  ;                                // start with uniform weights
2 for  $t = 1$  to  $T$  do
3   run  $\mathcal{A}$  on  $D$  with weights  $w_{ti}$  to produce a model  $M_t$ ;
4   calculate weighted error  $\epsilon_t$ ;
5   if  $\epsilon_t \geq 1/2$  then
6     | set  $T \leftarrow t - 1$  and break
7   end
8    $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$  ;                      // confidence for this model
9    $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$  for misclassified instances  $x_i \in D$  ;      // increase weight
10   $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$  for correctly classified instances  $x_j \in D$  ;    // decrease
11 end
12 return  $M(x) = \sum_{t=1}^T \alpha_t M_t(x)$ 
```



Figure 11.2, p.336

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 :

$$\frac{1}{2\epsilon_t} = \frac{\delta_t}{Z_t}$$

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

From this we can derive

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

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.

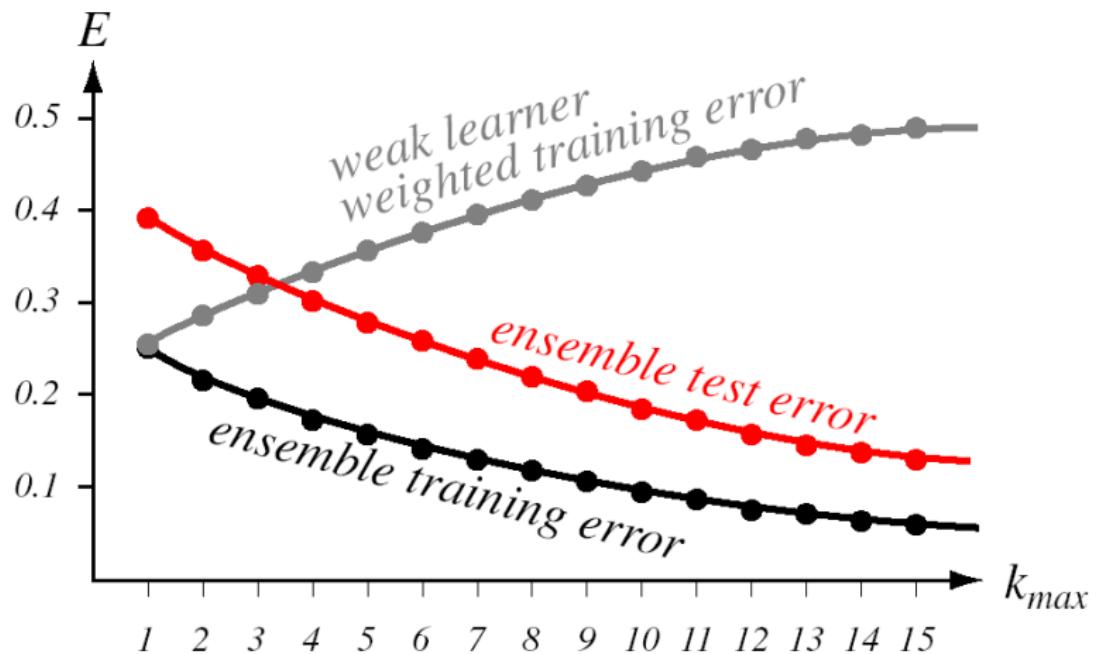
- focuses on “difficult” patterns

- 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

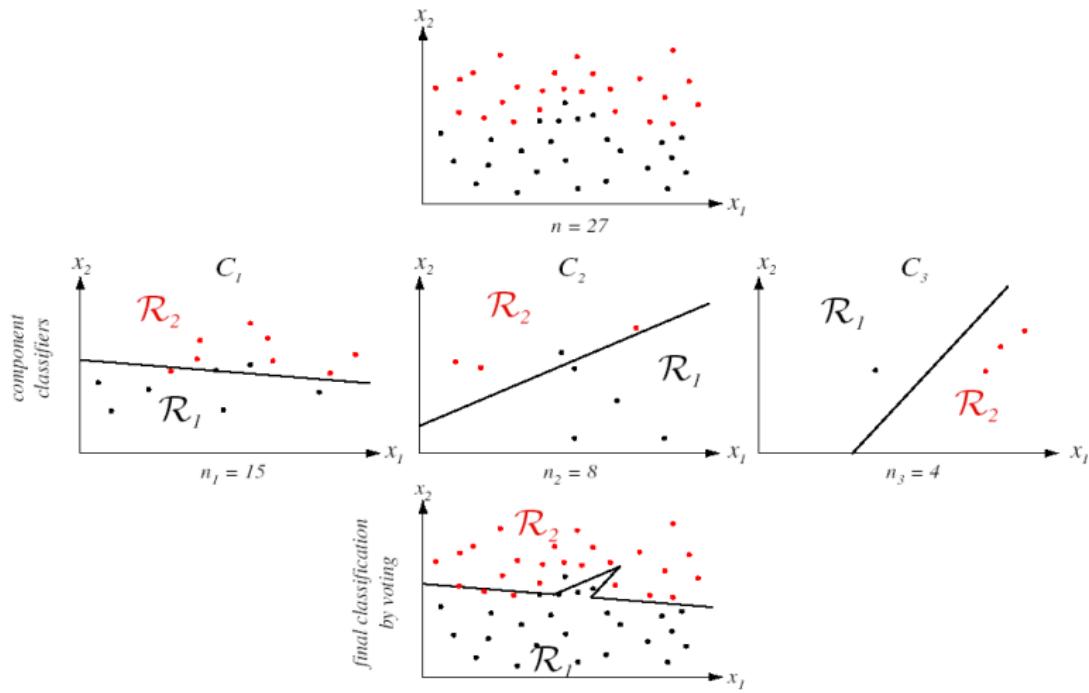
gives a non-linear decision boundary

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



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 ( *random forests* in [Algorithm 11.2](#)), whereas boosting is typically used with high-bias models such as linear classifiers or univariate decision trees (also called *decision stumps*).

Summary: Ensemble Learning

- ☞ No Free Lunch and conservation theorems → no “magic bullet” for machine learning
- ☞ Bias-variance decomposition breaks down error, illustrates 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