Ensemble Learning

COMP9417 Machine Learning and Data Mining

Term 2, 2023

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

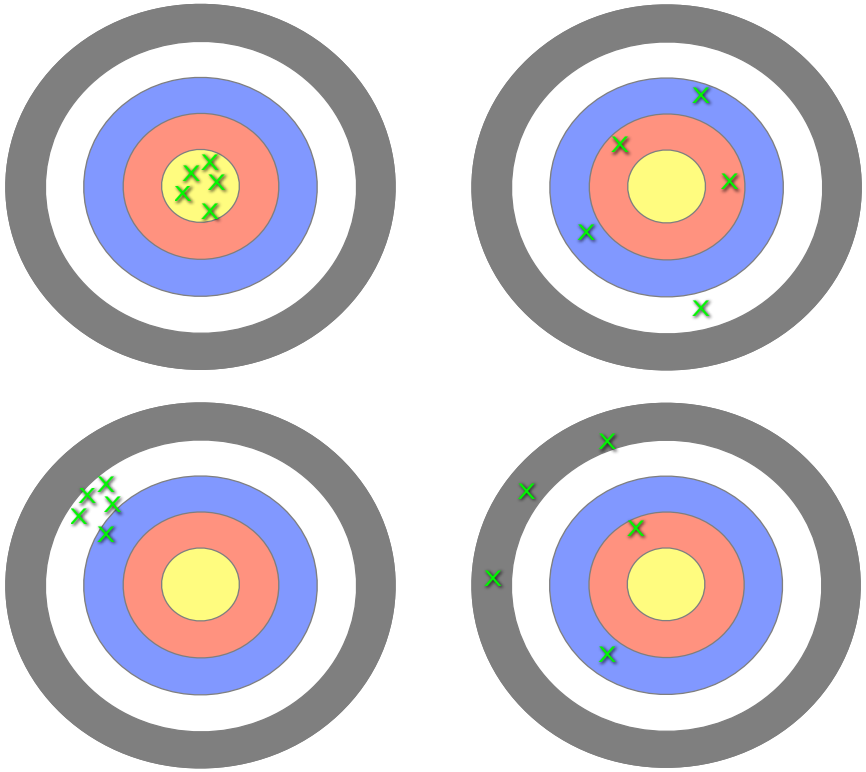
The bias-variance decomposition of error can be a tool for thinking about how to reduce error in learning — ensemble learning methods can be viewed from this perspective.

- Take a learning algorithm and ask:
  - how can we reduce its bias ?
  - how can we reduce its variance ?
- Ensemble learning methods can be seen a kind of *multi-level* learning:
  1. learn a number of base-level models from the data, and
  2. learn to combine these models as an ensemble
- How does the bias-variance decomposition of error relate to reducing error in learning using ensembles ?

The bias-variance decomposition

- A theoretical tool for analyzing how much any given training set affects performance of a learning algorithm
- Assume we have an infinite number of models trained by the same learning algorithm on different sample datasets, all of the same size:
  - The *bias* of a learning algorithm is the expected error due to the mismatch between the learner's model space and the space of target concepts
  - The *variance* of a learning algorithm is the expected error due to differences in the training datasets used
  - Total expected error $\approx$ bias^2 + variance
- Next slide: a graphical representation of this idea, where distance from "bullseye", i.e., centre of target stands for error

Bias and variance



Each target corresponds to a different learning *algorithm*, each arrow (green 'x') a different training *sample*. Top row learning algorithms show low bias, on average close to the centre (i.e., the true function value), while on bottom row they have high bias. Left column algorithms show low variance and, in the right column, high variance.

Bias-variance: a trade-off

Bias-variance components of error typically show a trade-off:

Bias $\uparrow$ and Variance $\downarrow$, or

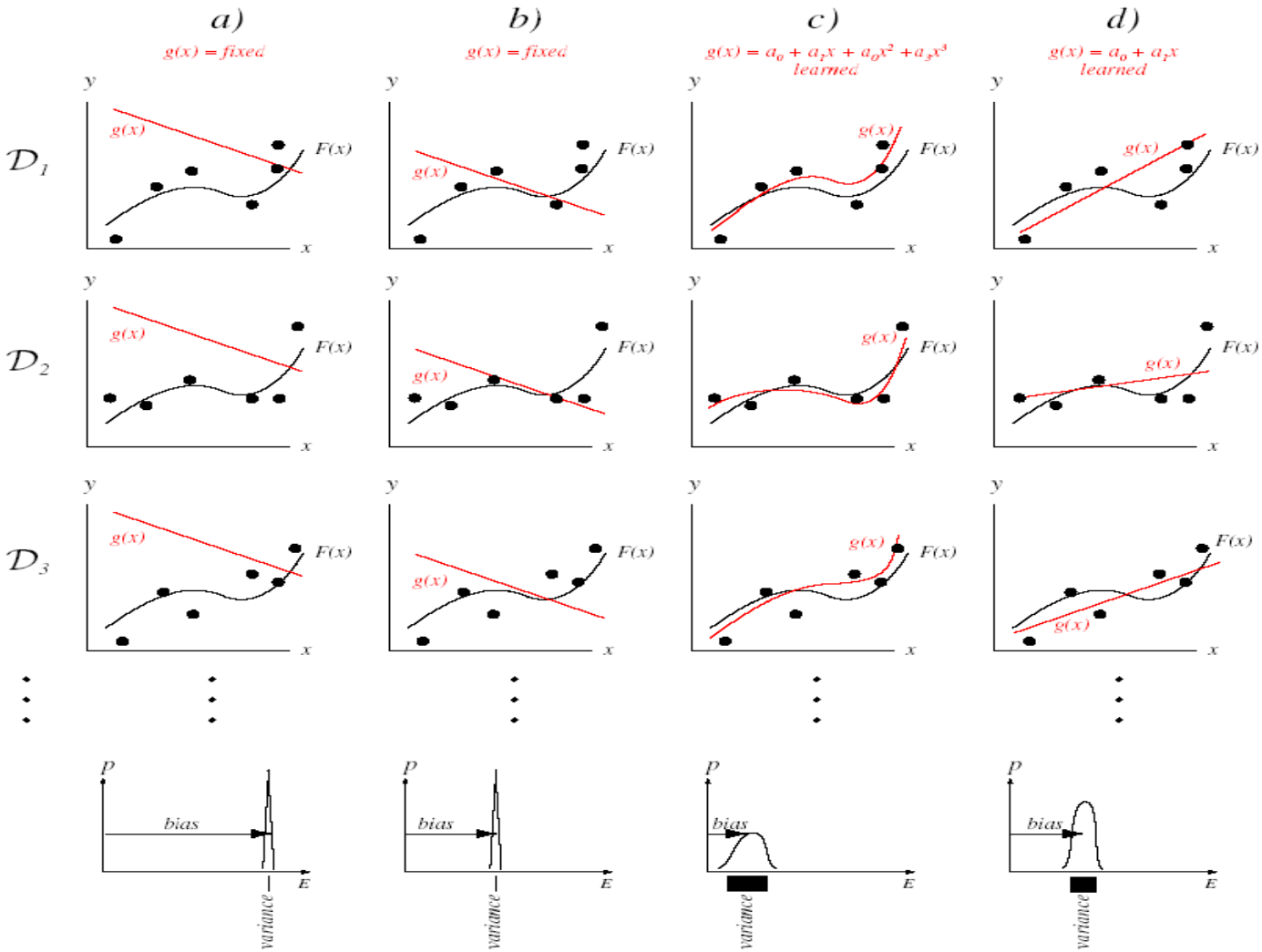
Bias $\downarrow$ and Variance $\uparrow$

This trade-off shown with regression in the following figure [1] (to see the details you may have to zoom in in your viewer):

- each column represents a different regression model class $g(x)$ shown in red
- each row represents a different set of $n = 6$ training points, D_i, randomly sampled from target function $F(x)$ with noise, shown in black
- probability functions of mean squared error E are shown

[1] See: Duda et al. (2001).

Bias-variance: a trade-off



Bias-variance: a trade-off

Previous slide:

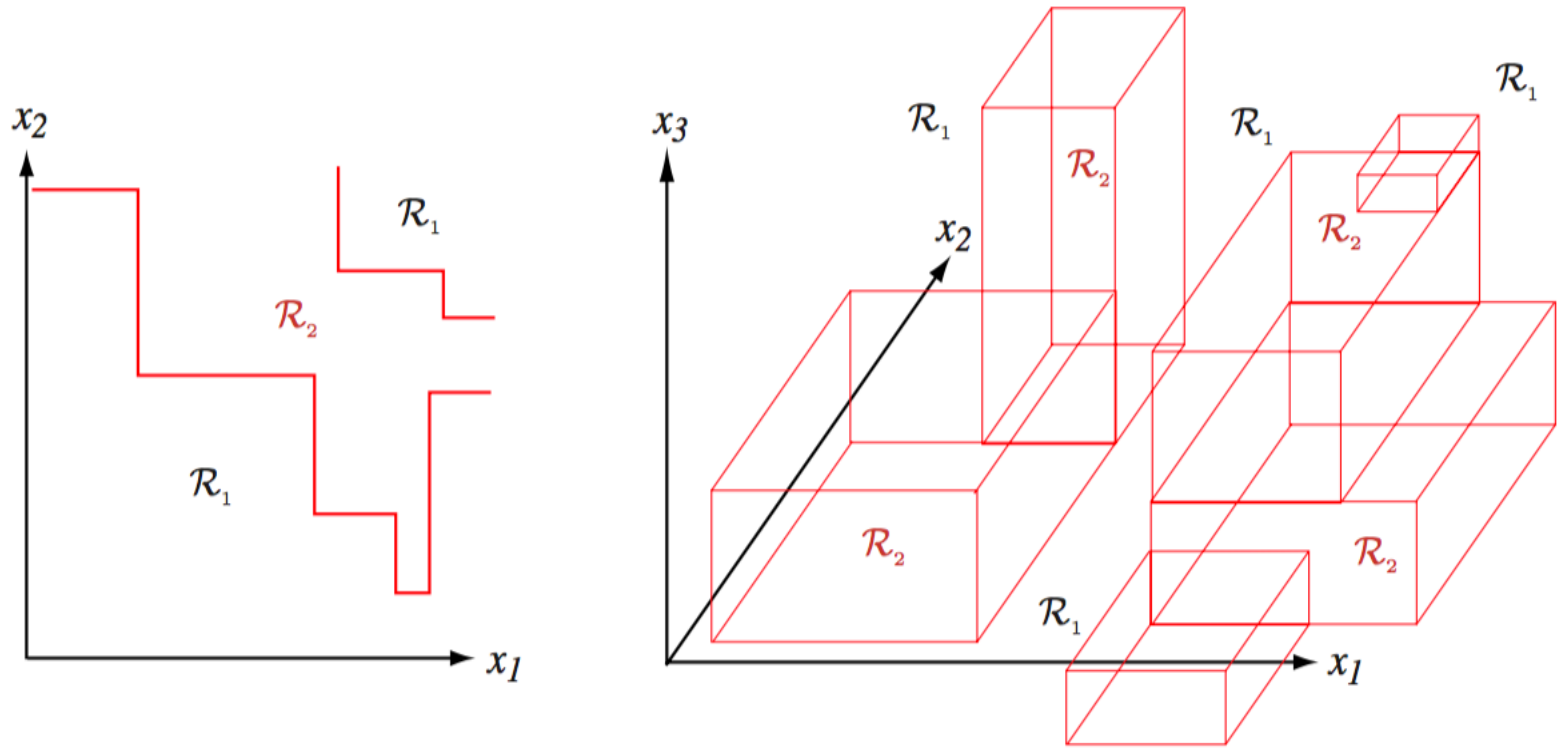
- $a)$ is very poor: a linear model with fixed parameters independent of training data; high bias, zero variance
- $b)$ is better: a linear model with fixed parameters independent of training data; slightly lower bias, zero variance
- $c)$ is a cubic model with parameters trained by mean-square-error on training data; low bias, moderate variance
- $d)$ is a linear model with parameters adjusted to fit each training set; intermediate bias and variance
- training with data $n \to \infty$ would let
  - $c)$ tend to bias approaching small value due to noise,
  - but this bias-reduction would not be the case for $d)$,
  - and variance for all models would approach zero.

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"*

Decision boundaries in tree learning



Decision boundaries for monothetic two-class trees in two and three dimensions; arbitrarily fine decision regions for classes $\mathcal{R}_1$, $\mathcal{R}_2$ can be learned by recursively partioning the instance space.

From: Duda et al. (2001).

Instability of tree learning

An example shows the effect of a small change in the training data on the structure of an unpruned binary tree learned by CART. The training set has 8 instances for each class:

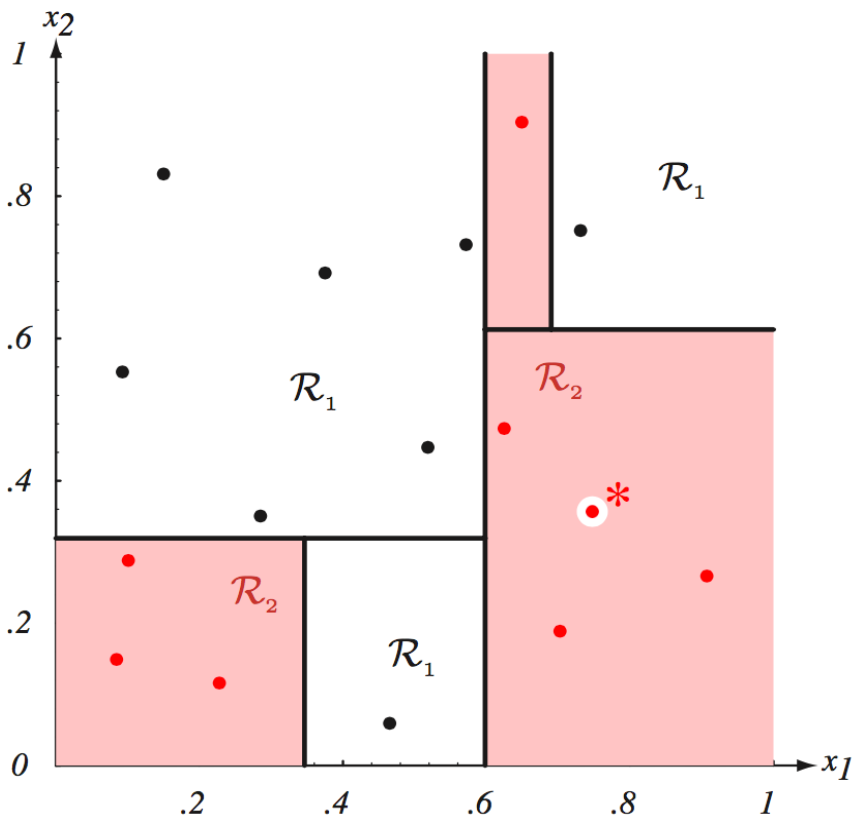
ω_1 (black)	
x_1	x_2
.15	.83
.09	.55
.29	.35
.38	.70
.52	.48
.57	.73
.73	.75
.47	.06

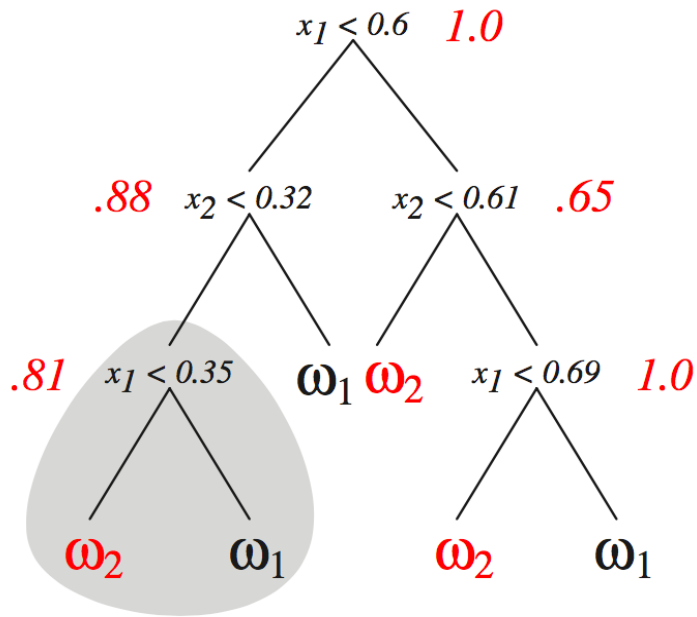
ω_2 (red)	
x_1	x_2
.10	.29
.08	.15
.23	.16
.70	.19
.62	.47
.91	.27
.65	.90
.75	.36* (.32†)

Note: for class ω_2 (red) the last instance has two values for feature x_2. On the next slide is a tree learned from the data where this instance has value $x_2 = .36$ (marked *), and on the following slide we see the tree obtained when this value is changed to $x_2 = .32$ (marked †).

From: Duda et al. (2001).

Instability of tree learning

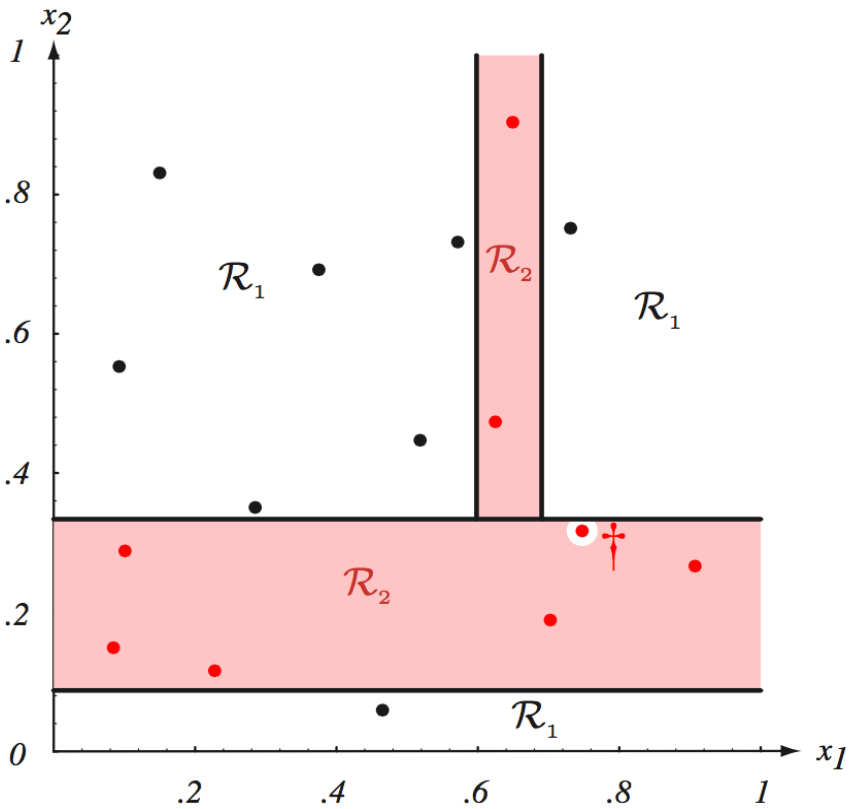


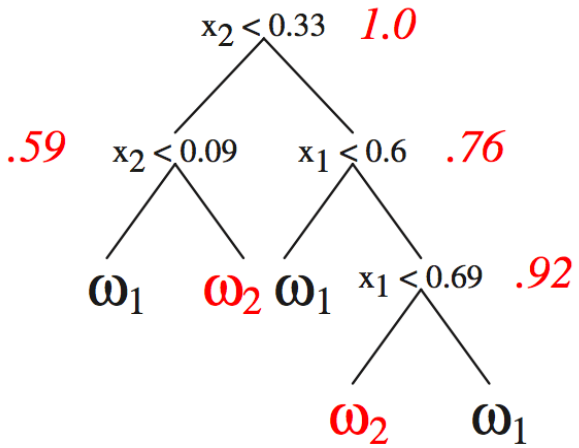


The partioned instance space (left) contains the instance marked * and corresponds to the decision tree (right).

From: Duda et al. (2001).

Instability of tree learning





The partioned instance space (left) contains the instance marked † and corresponds to the decision tree (right). Note that both the decision boundaries and the tree topology are considerably changed, for example, testing x_2 rather than x_1 at the tree root, although the change in data was very small.

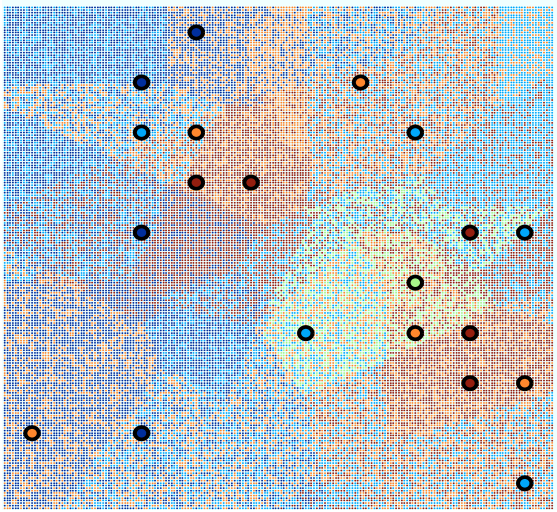
From: Duda et al. (2001).

Stability and Bias-Variance

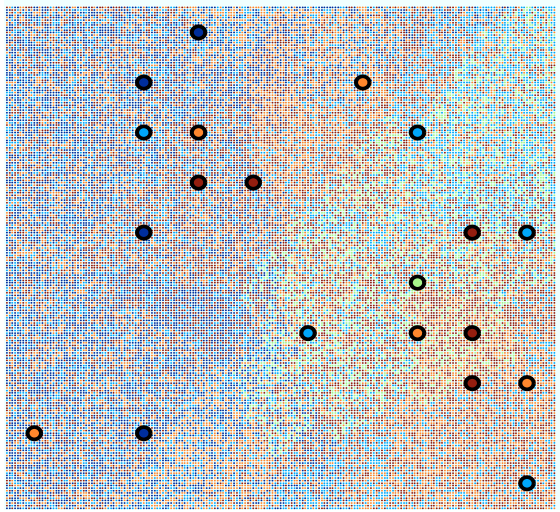
- stable algorithms typically have high bias
- unstable algorithms typically have high variance
- BUT: take care to consider effect of parameters
  - e.g., decision trees typically have parameters to limit depth of tree
  - simple trees will be more stable
- also, e.g., in kNN
  - 1NN perfectly separates 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

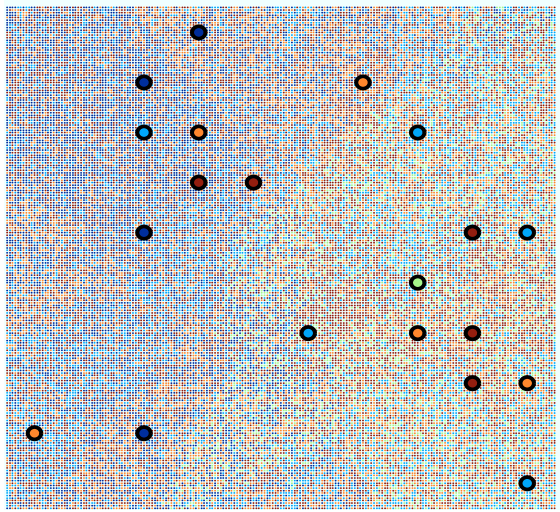
Decision regions of k-nearest neighbour classifiers on a multiclass classification problem; shading represents the predicted probability distribution over the five classes.



3-nearest neighbour



5-nearest neighbour



7-nearest neighbour

As k increases, decision boundaries become less sharp (bias $\uparrow$, variance $\downarrow$).

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 combine their output predictions
- 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

Bias-variance in ensemble classification

- Recall that we originally derived the bias-variance decomposition for regression – squared-error loss function
- Cannot apply same derivation for classification – zero-one loss
- Bias-variance decomposition used to analyze how much restriction to a single training set affects performance
- Can decompose expected error of any individual ensemble member as follows:
  - Bias = expected error of the ensemble classifier on new data
  - Variance = component of the expected error due to particular training set being used to build classifier
  - Total expected error $\approx$ bias^2 + variance
- Note (A): we assume noise inherent in the data is part of the bias component as it cannot normally be measured
- Note (B): multiple versions of this decomposition exist for zero-one loss but the basic idea is always the same

Bagging

"**B**ootstrap **Agg**regation" – introduced by Leo Breiman[2]

- 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., high-variance, e.g., decision trees)

[2] See: Breiman (1996).

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, with diminishing returns
- Problem: we only have one dataset of n instances !
- Solution: generate multiple samples from the dataset
  - use sampling *with replacement* from original dataset
  - each sample can have duplicate examples
  - each sample can have missing examples
- Can be applied to numeric prediction and classification

Bagging more precisely

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

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

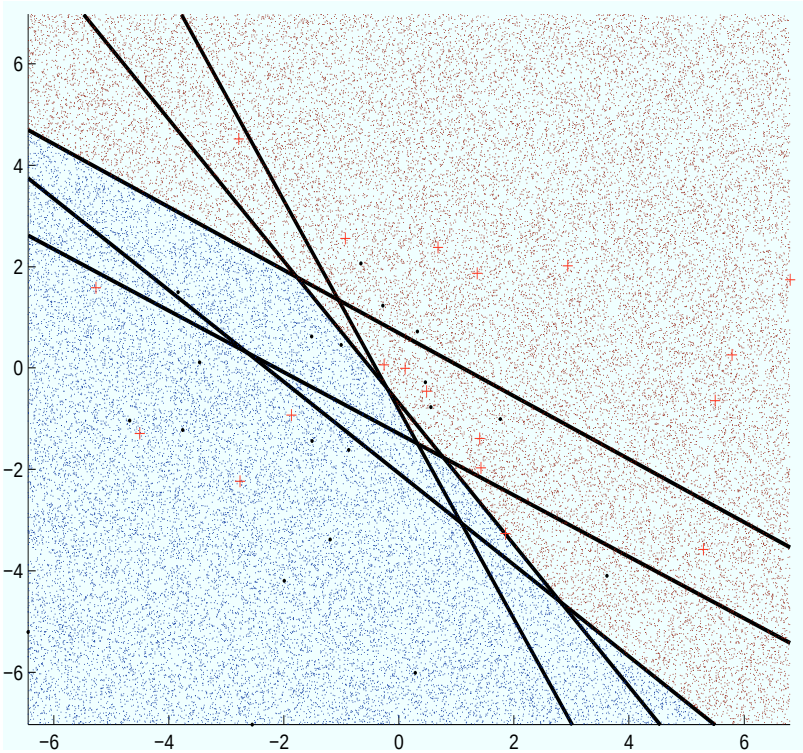
Output: set of models; predictions to be combined by voting or averaging.

```
for t = 1 to T do
    bootstrap sample D_t from D by sampling |D| examples with replacement
    run A on D_t to produce a model M_t
end
return {M_t | 1 ≤ t ≤ T}
```

What is the ensemble's prediction ? The *average* of the predictions of all the models in the ensemble:

- for classification, the majority vote, or mode
- for regression, the mean

Bagging linear classifiers



An ensemble of five *basic linear classifiers* built from bootstrap samples with bagging.

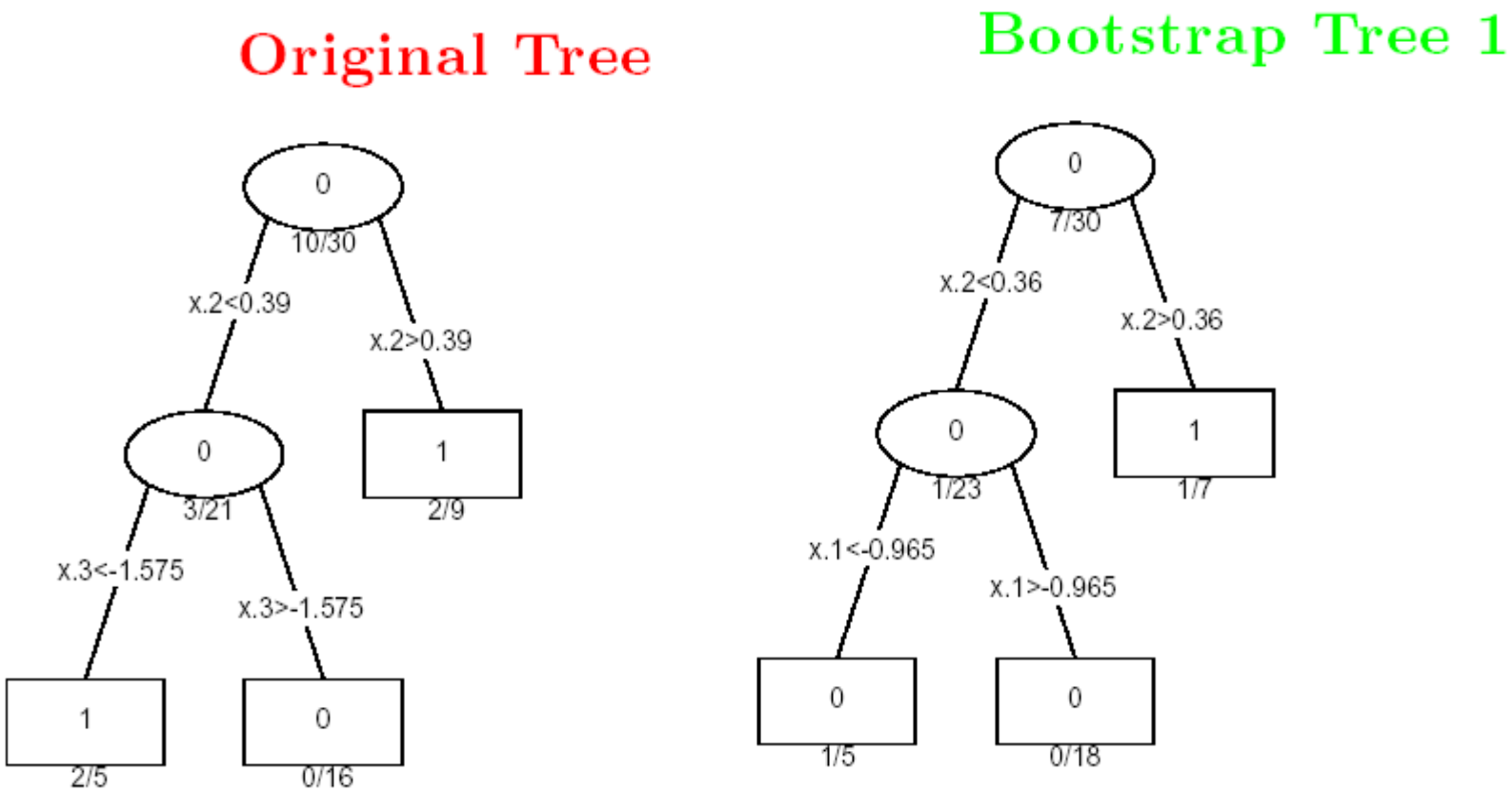
The decision rule is majority vote, leading to a piecewise linear decision boundary.

Bagging trees

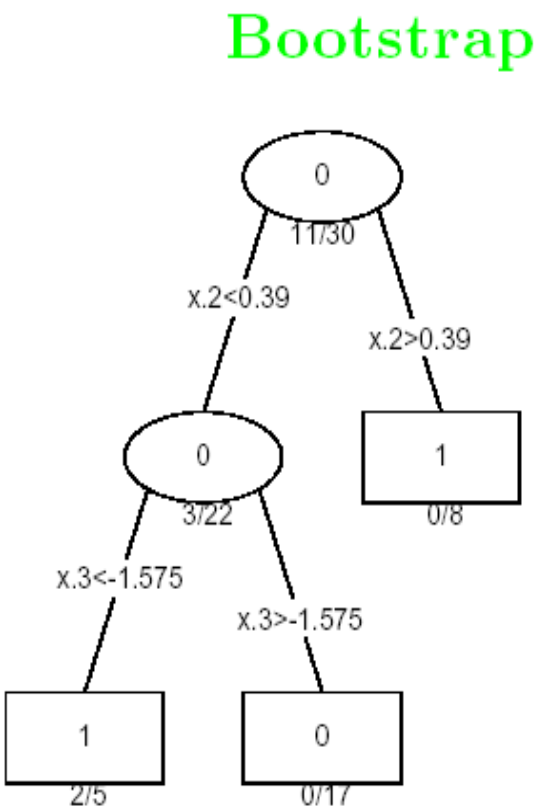
An experiment with simulated data:

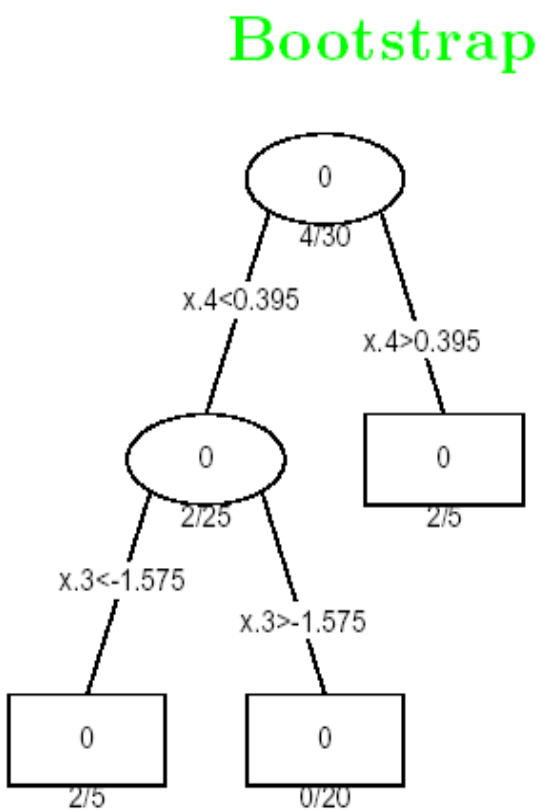
- sample of size $n = 30$, two classes, five features
- $Pr(Y = 1|x_1 \leq 0.5) = 0.2$ and $Pr(Y = 1|x_1 > 0.5) = 0.8)$
- test sample of size 2000 from same population
- fit classification trees to training sample, 200 bootstrap samples
- trees are different (tree induction is *unstable*)
- therefore 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

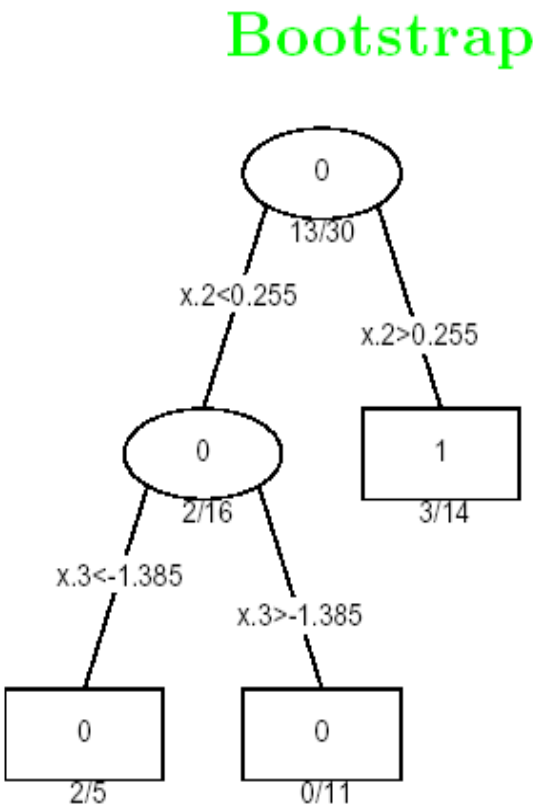


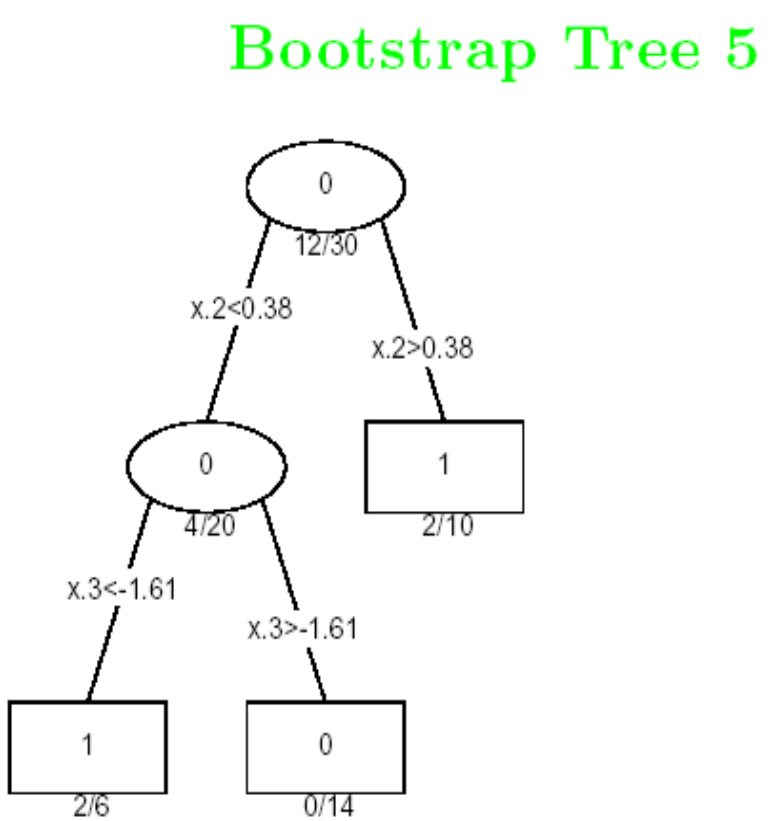
Bagging trees



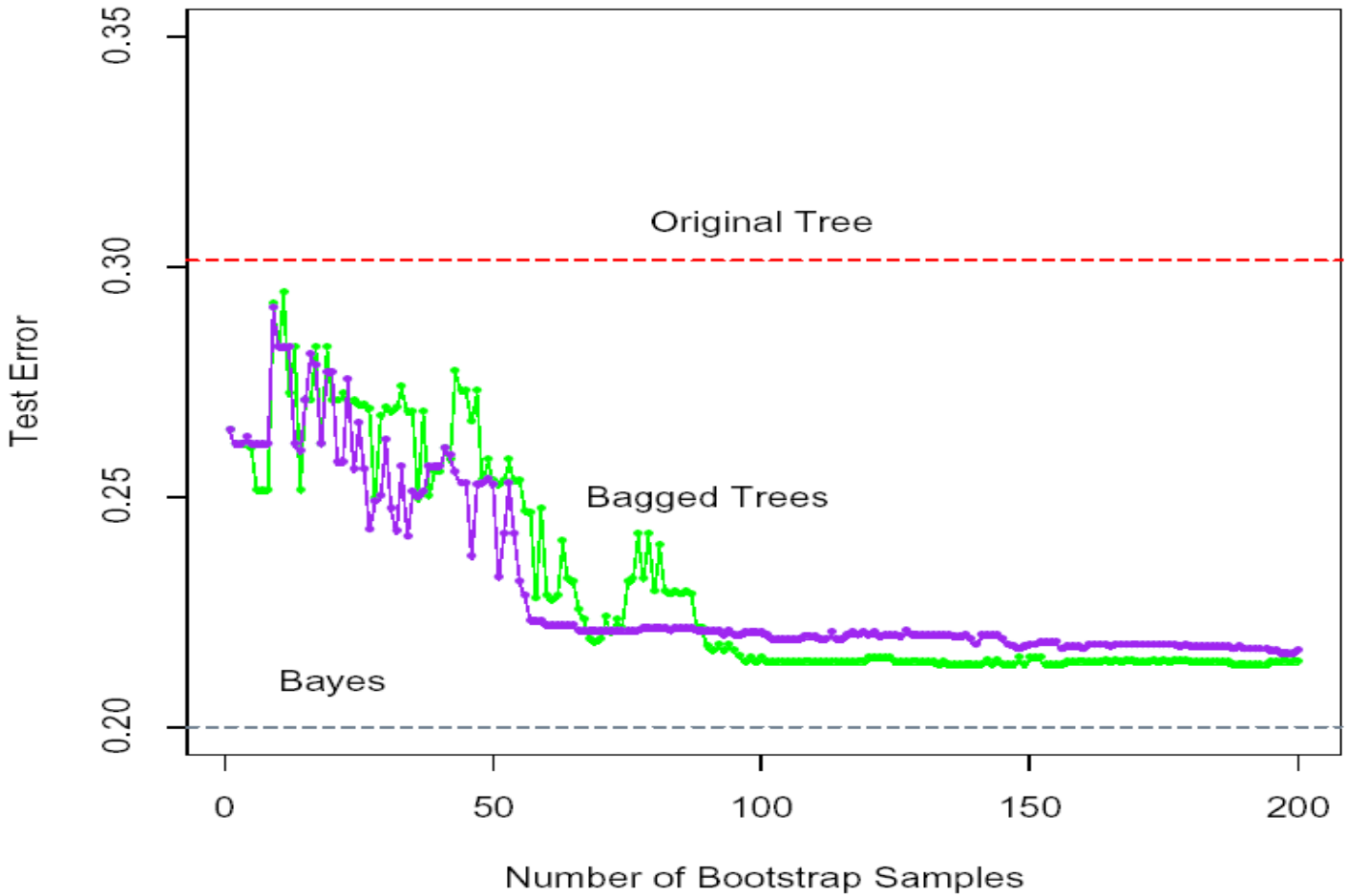


Bagging trees





Bagging trees



Bagging trees

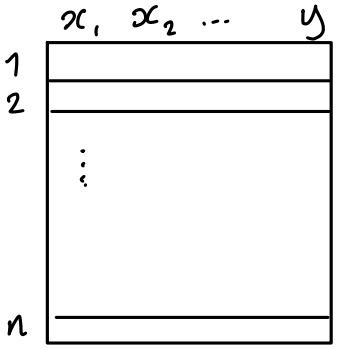
The news is not all good:

- 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

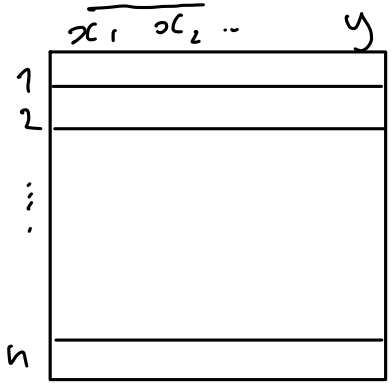
Bagging

p features
n examples

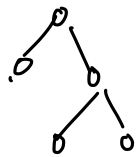


typically $d < p$

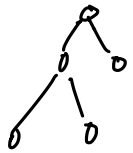
RF



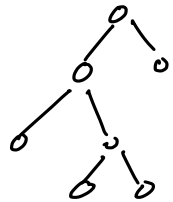
Model 1



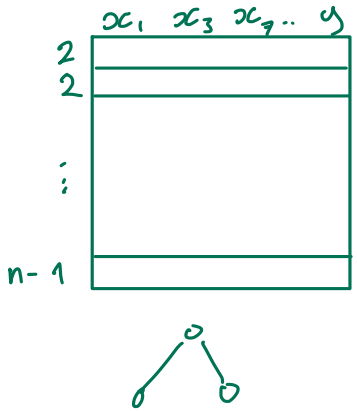
Model 2 ...



Model T

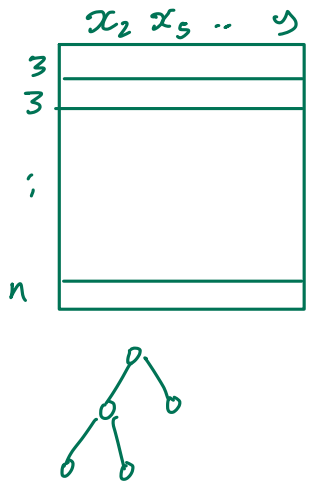


Train 1:
random sample w/ replacement of size n
select d features



Model 1

Train 2:
random sample w/ replacement of size n
select d features



Model ..

Question:
what about bad selection of features?

Answer:
because of majority voting for output, may not matter

Model T

Randomization

- Can randomize learning algorithm instead of input to introduce diversity into an ensemble
- Some learning algorithms already have a random component
- Most algorithms can be randomized, e.g., greedy algorithms:
  - Pick r options at random from the full set of options, then choose the best of those r choices
  - E.g., feature selection in decision trees
- More generally applicable than bagging: e.g., we can use random subsets of features in a nearest-neighbor classifier
  - Bagging does not work well with stable classifiers
- Randomization can be combined with bagging
  - When learning decision trees, this yields the Random Forest[3] method for building ensemble classifiers

[3] See: Breiman (2001).

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 that 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

- Boosting[4] also uses voting/averaging of an ensemble but each model is *weighted* according to their performance
- An iterative learning 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
  - Intuition: models should be experts that complement each other
- There are *many* variants of this method ...

[4] See: Freund and Schapire (1997).

The strength of weak learnability

Setting comes from computational learning theory

- 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 !
- a **strong** learner if $\epsilon < 0.5$ and ϵ "close" to zero.
- a **weak** learner if $\epsilon < 0.5$ and ϵ "close" to 0.5.
  - Question: is there a procedure to convert a weak learner into a strong learner ?
  - Answer: yes, weak learners can be *boosted* into strong learners !

Weight updates in boosting

- Suppose a *linear classifier* achieves performance as in the contingency table below. Error rate is $\epsilon = (9 + 16)/100 = 0.25$.

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

- 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
  - a factor $1/2(1 - \epsilon) = 2/3$ for the correctly classified examples.

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 datasets

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
  if $\epsilon_t \geq 1/2$ **then**
    set $T \leftarrow t - 1$ and break
  end
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
  $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ 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 $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$

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

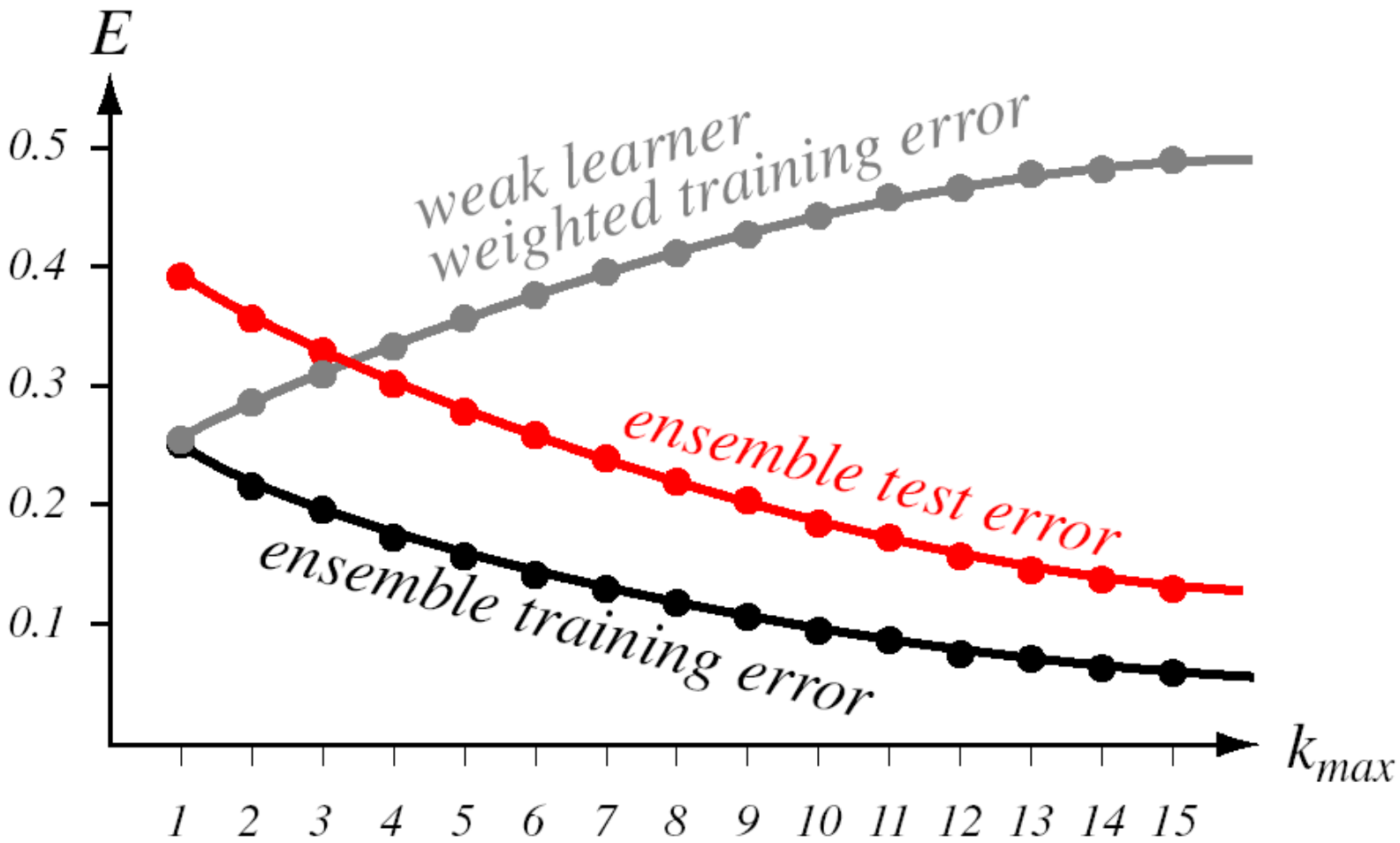
$$w_{(t+1)i} = w_{ti}\frac{\exp(-\alpha_t y_i M_t(x_i))}{Z_t}$$

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

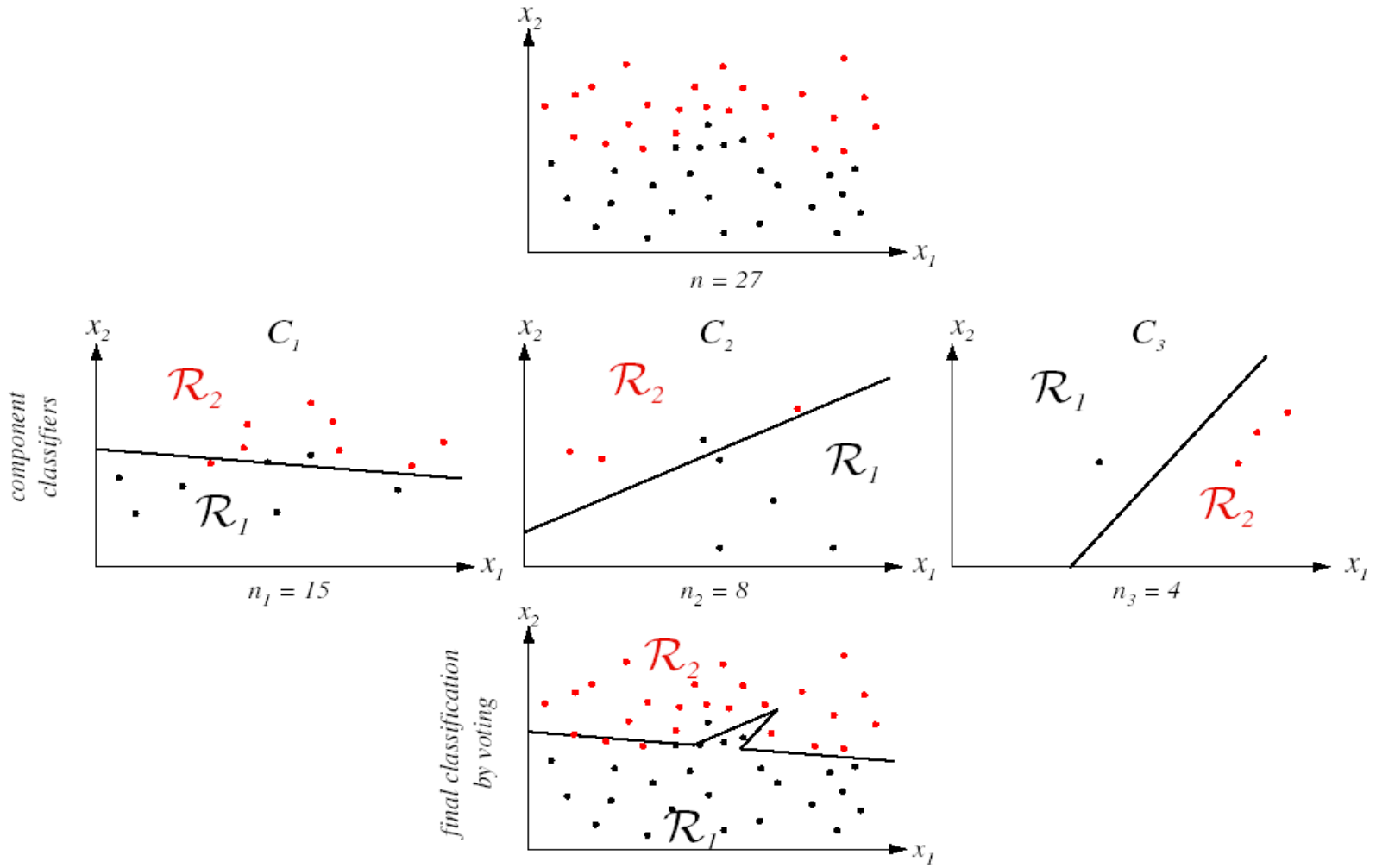


Boosting enlarges the model class

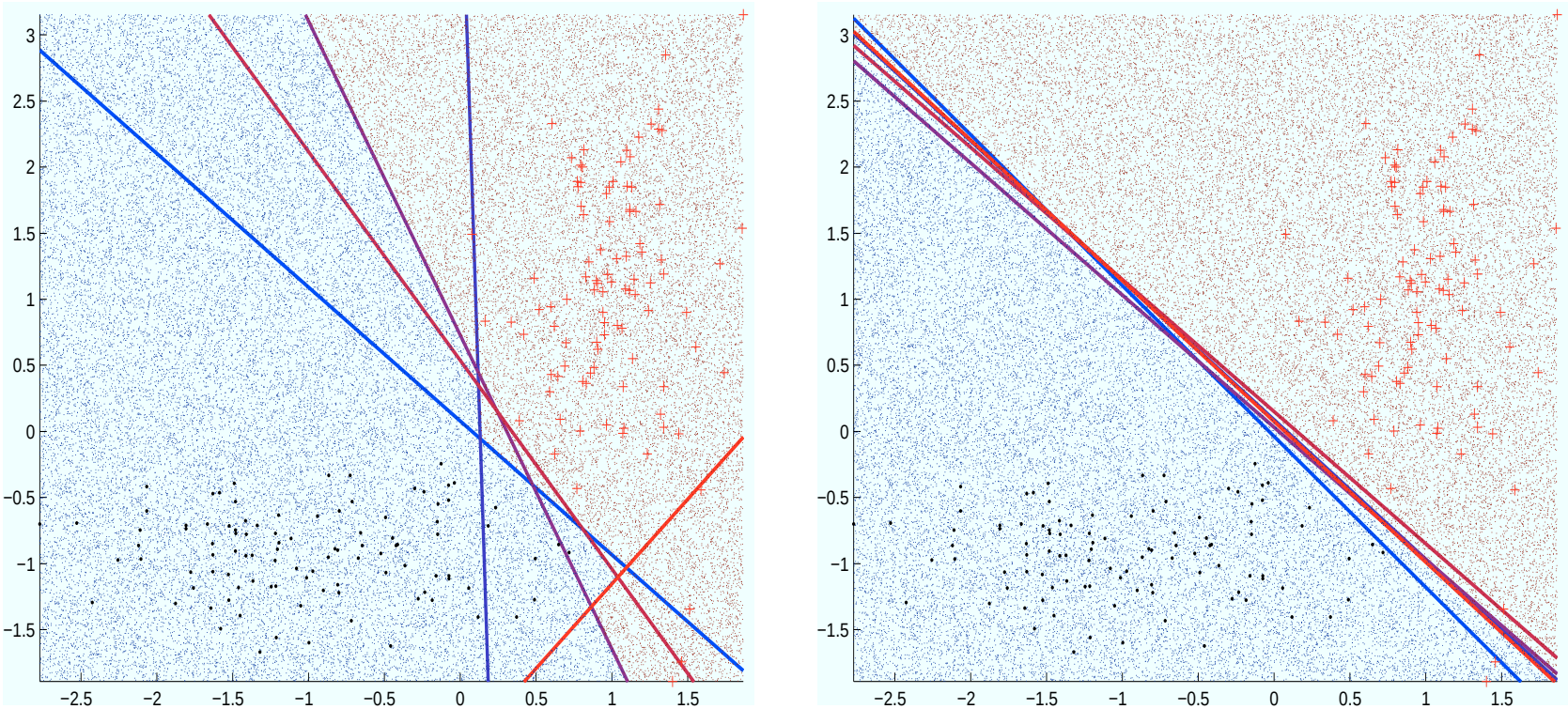
A two-dimensional two-category classification task

- three component linear classifiers
- final classification is by weighted voting of 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

Boosting enlarges the model class



Boosting vs. Bagging



(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.

Stacking

- So far, ensembles where base learners all use same algorithm
- But what if we want to combine outputs of different algorithms ?
- Also, what if the combining method could be tuned from data ?
- "Stacked generalization" or stacking
- Uses *meta learner* instead of voting to combine predictions of base learners
  - Predictions of base learners (level-0 models) are used as input for meta learner (level-1 model)
- Each base learner considered a feature, with value its output $\hat{y}$ on instance x
- But predictions on training data can't be used to generate data for level-1 model!
  - So a cross-validation-like scheme is employed

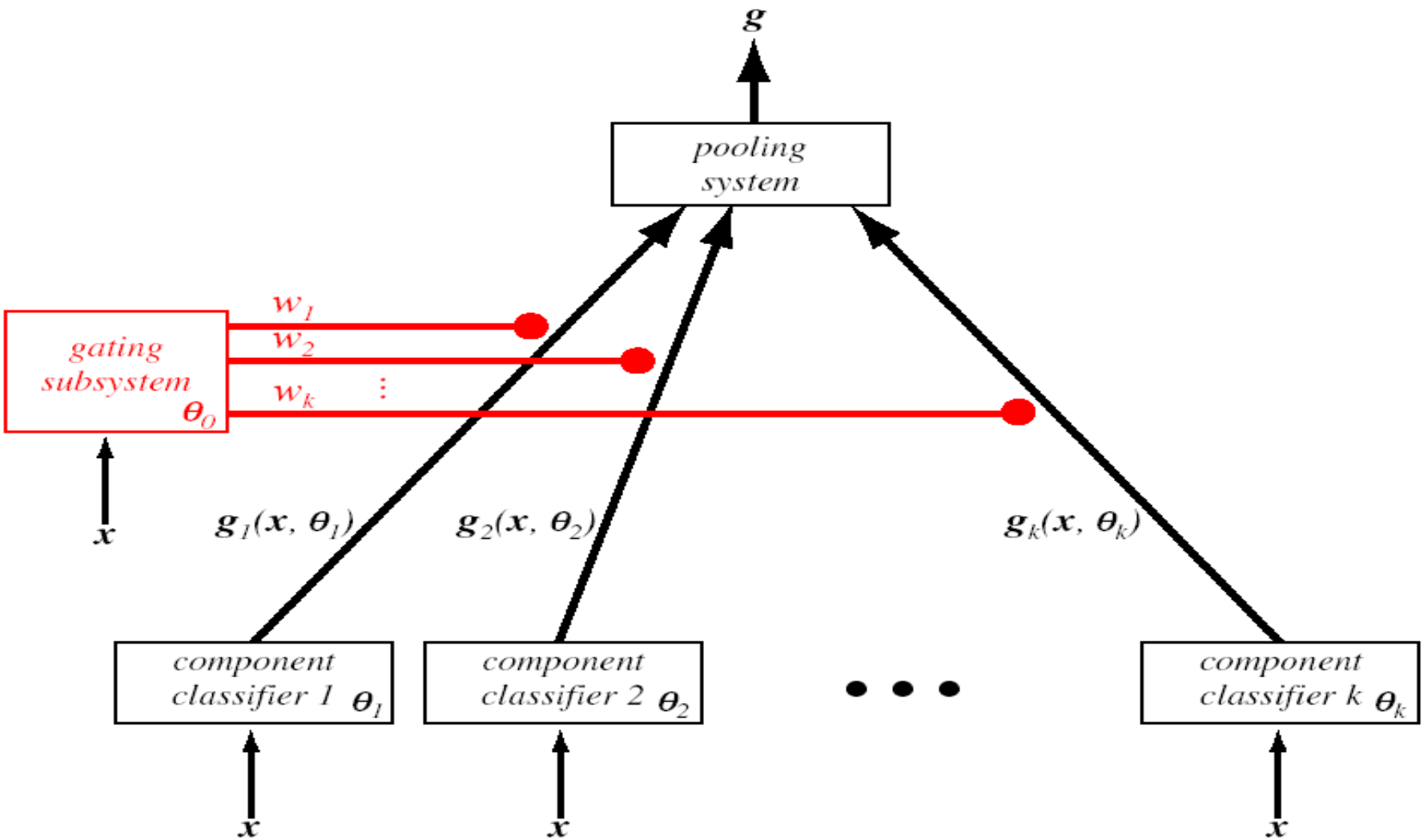
Stacking

- If base learners can output probabilities it's better to use those as input to meta-learner
  - gives more information to meta-learner
- Which algorithm to use to generate meta-learner?
  - In principle, any learning scheme can be applied, but suggested to use
  - "relatively global, smooth" models (David Wolpert)
  - A linear model is typically used
    - Since base learners do most of the work
    - And this reduces risk of overfitting
- Stacking can also be applied to numeric prediction (and density estimation)

Mixture of Experts

- Framework for learning assuming data generated by a *mixture model*
  - base level component classifiers (or rankers, . . .)
  - outputs are combined by a tunable system to do the "mixing"
- Each component models an "expert" for some part of the problem
- All component outputs are pooled for ensemble output
- Can be trained by gradient descent

Mixture of Experts



Additive Regression

- Using statistical terminology, boosting is a greedy algorithm for fitting an additive model
- More specifically, it implements forward stagewise additive modeling
- Forward stagewise additive modeling for numeric prediction:
  1. Build standard regression model (e.g., regression tree)
  2. Gather residuals, learn model predicting residuals (e.g. another regression tree), and repeat
- To predict, simply sum up individual predictions from all regression models

Gradient (Tree) Boosting is based on this approach, where at each boosting iteration a model is fit to approximate the components of the *negative gradient* of the overall loss Friedman (2001).

Gradient Boosting

- Boosting algorithms learn a form of *additive model*
  - training uses a *forward stagewise* procedure
- At each boosting iteration, a new (weighted) component function is added to the boosted model
- In *gradient boosting*, this approach is used to solve an optimization problem
  - Informally, need to minimize loss over all components (basis functions) over all training examples
- A simpler approximation to this optimization is a forward stepwise procedure
  - At each iteration, minimize loss summed over all previously added components, plus the current one
  - In gradient boosting, a regression tree is learned at each iteration to minimize the loss for predicting the *negative gradient* at each leaf
- Implemented in the widely-used XGBoost[5] package for scalable learning

[5] See: Chen and Guestrin (2016).

Forward Stagewise Additive Modelling

initial model : $f_0(x)$

for $t = 1, \ldots, T$ do

(i) $(\alpha_t, \beta_t) = \arg\min_{\alpha, \beta} \sum_{i=1}^{N} L(y_i, f_{t-1}(x_i)) + \alpha\, b(x_i; \beta)$

(ii) set $f_t(x) = f_{t-1}(x) + \alpha_t\, b(x; \beta_t)$

where $b(\,;\,)$ is a base model.

Gradient Boosting (based on numerical optimization)

- at each stage, fit a tree T to approximate negative gradient:

$$\theta_t = \arg\min_{\theta} \sum_{i=1}^{N} (-g_{it} - T(x_i; \theta))^2$$

for some differentiable loss function L (regression, classification).

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 & randomization approach for trees
- Boosting has a more theoretically justified basis and often works better in practice to reduce error, but can be susceptible to very noisy data
- Many other variants of ensemble learning
- Gradient Boosting (XGBoost, LightGBM, CatBoost) often the "off-the-shelf" method of choice

Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (2001). Random Forests. *Machine Learning*, 45:5–32.

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.

Duda, R., Hart., P., and Stork, D. (2001). *Pattern Classfication*. Wiley.

Freund, Y. and Schapire, R. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Friedman, J. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232.