TREE-BASED METHODS

(Part 2)

Bagging, Random Forest, Boosting, and Stacking

盧信銘 台大資管系

BAGGING

Bagging

- Bootstrap aggregation or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method.
- Often used in the context of decision trees.
- Bagging can be applied in almost all other learning models.
- Recall that given a set of n independent observations, $Z_1, Z_2, \ldots, Z_n$, each variance σ^2, the variance of the mean $\bar{Z}$ of the observation is σ^2/n.
- That is, averaging a set of observations reduces variance.
- Problem: we usually do not have access to multiple training sets.

Bagging (Cont'd.)

- Idea: We can do bootstrap.
- Bootstrap: Take repeated samples from the (single) training data set.
- ➔ Random sample with replacement. The new dataset has the same number of observations as the original training dataset.
- Example: Original dataset = {1,2,3,4,5,6,7,8,9,10}
- Bootstrap dataset1 = {10,7,6,5,9,1,7,2,7,8}
- Bootstrap dataset2 = {6,9,10,1,10,6,7,10,2,7}
- …

- Note: It is possible to have duplicated data points in boostrapped dataset.

Bagging (Cont'd.)

- We can generate B different bootstrapped training data sets.
- Train our method on the b-th bootstrapped training set to get $\hat{f}^{*b}(x)$, the prediction at a point x.
- We then average all the predictions to obtain
- $\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$.

- For classification: majority vote among all B trees
- This is called bagging

Bagging (Tree-Based Models)

- Note: When applied to tree-based models, these trees are not pruned, so each individual tree has high variance but low bias. Averaging these trees reduces variance, and thus we end up lowering both variance and bias ☺

- For classification prediction, there are two approaches:
  1. Record the class that each bootstrapped data set predicts and provide an overall prediction to the most commonly occurring one (majority vote).
  2. If our classifier produces probability estimates we can just average the probabilities and then predict to the class with the highest probability.
- Both methods work well.

Bagging classifiers

Classifier generation

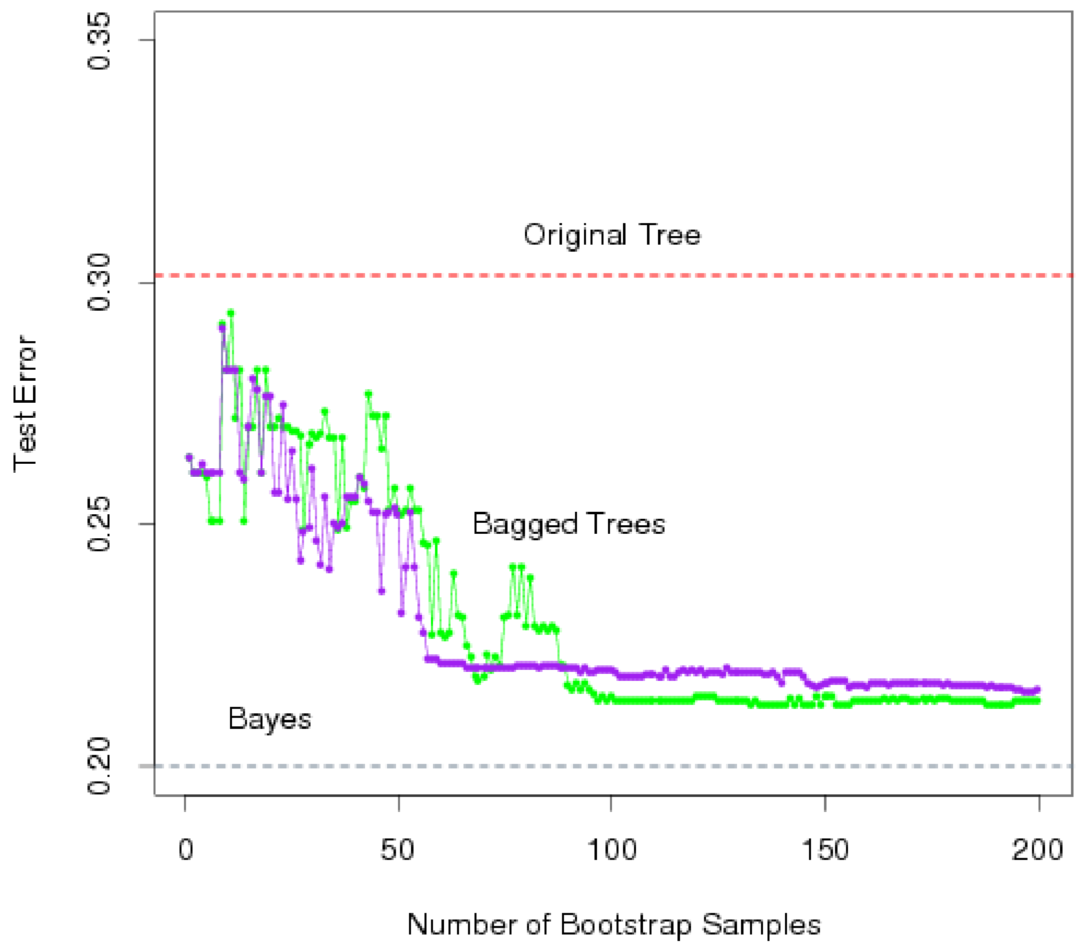
```
Let $n$ be the size of the training set.
For each of $t$ iterations:
   Sample $n$ instances with replacement from the
   training set.
   Apply the learning algorithm to the sample.
   Store the resulting classifier.
```

classification

```
For each of the $t$ classifiers:
   Predict class of instance using classifier.
Return class that was predicted most often.
```

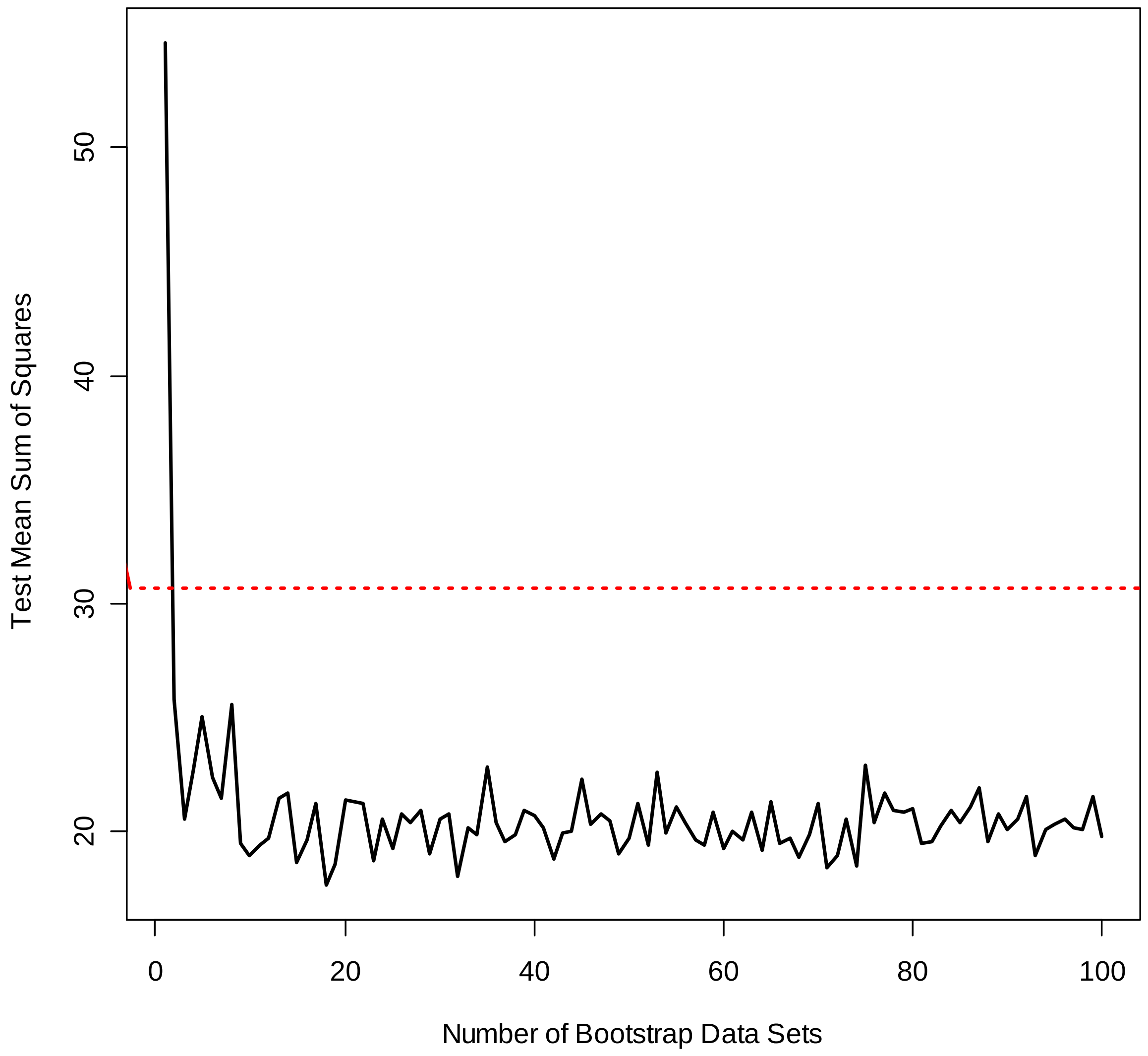
A Comparison of Error Rates

- Here the green line represents a simple majority vote approach
- The purple line corresponds to averaging the probability estimates.
- Both do far better than a single tree (dashed red) and get close to the Bayes error rate (dashed grey).



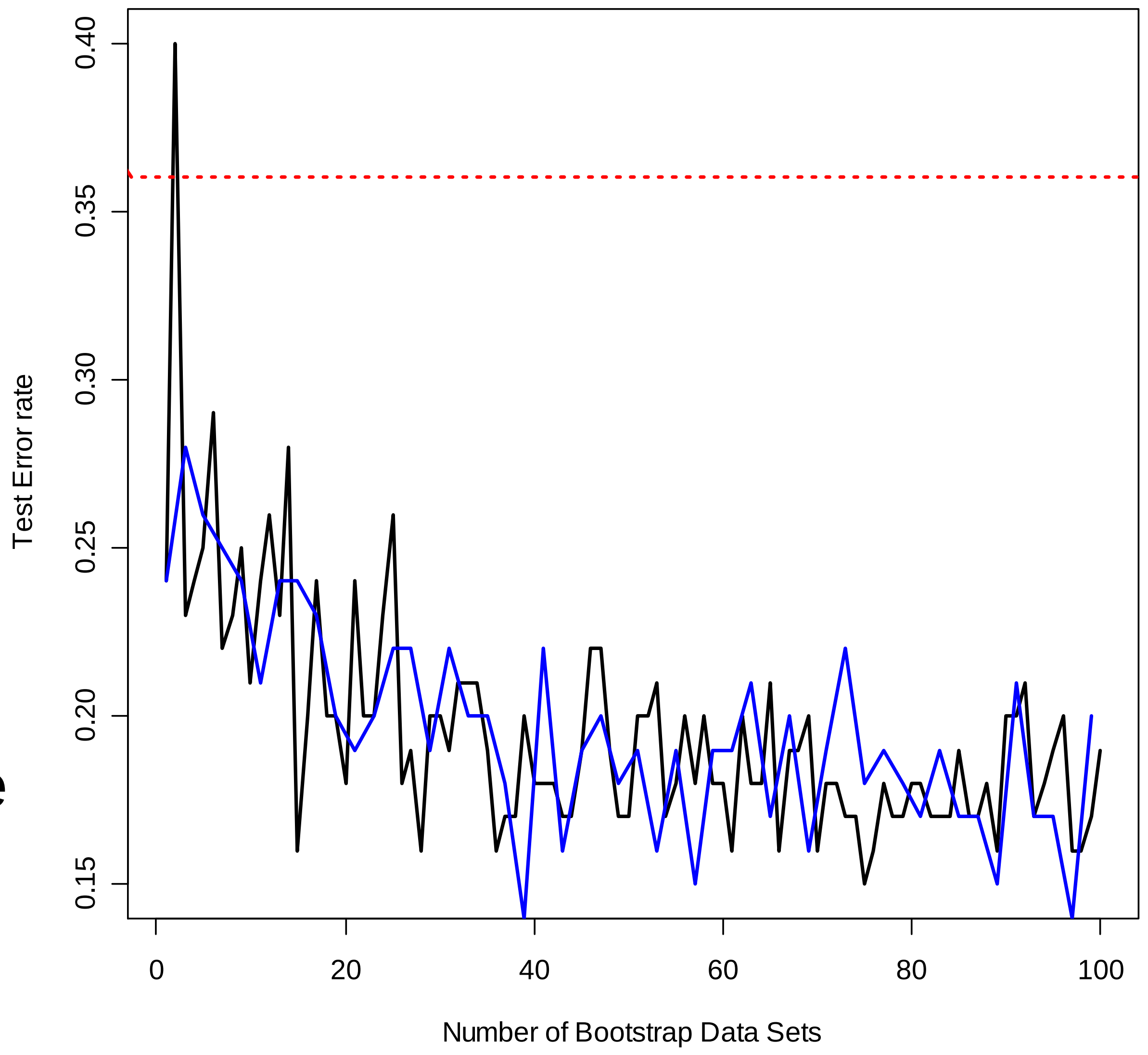
Example 1: Housing Data

- The red line represents the test mean sum of squares using a single tree.
- The black line corresponds to the bagging error rate



Example 2: Car Seat Data

- The red line represents the test error rate using a single tree.
- The black line corresponds to the bagging error rate using majority vote while the blue line averages the probabilities.



Out-of-Bag Error Estimation

- Since bootstrapping involves random selection of subsets of observations to build a training data set, then the remaining non-selected part could be the testing data.
- On average, each bagged tree makes use of around 2/3 of the observations, so we end up having 1/3 of the observations used for testing

RANDOM FORESTS

Random Forest

- Random forests provide an improvement over bagged trees by decorrelates the tree. ➔ Reduce the variance when averaging the tree.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors.
  - The split is allowed to use only one of these m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$.
- The number of predictors considered at each split is approximately equal to the square root of the total number of predictors
  - E.g for a dataset with 13 variables, randomly select 4 variables at each split.

Random Forest

Classifier generation

```
Let $n$ be the size of the training set.
For each of $t$ iterations:
   (1) Sample $n$ instances with replacement from
  the training set.
   (2) Learn a decision tree s.t. the variable
  for any new node is the best variable among $m$
  randomly selected variables.
   (3) Store the resulting decision tree.
```

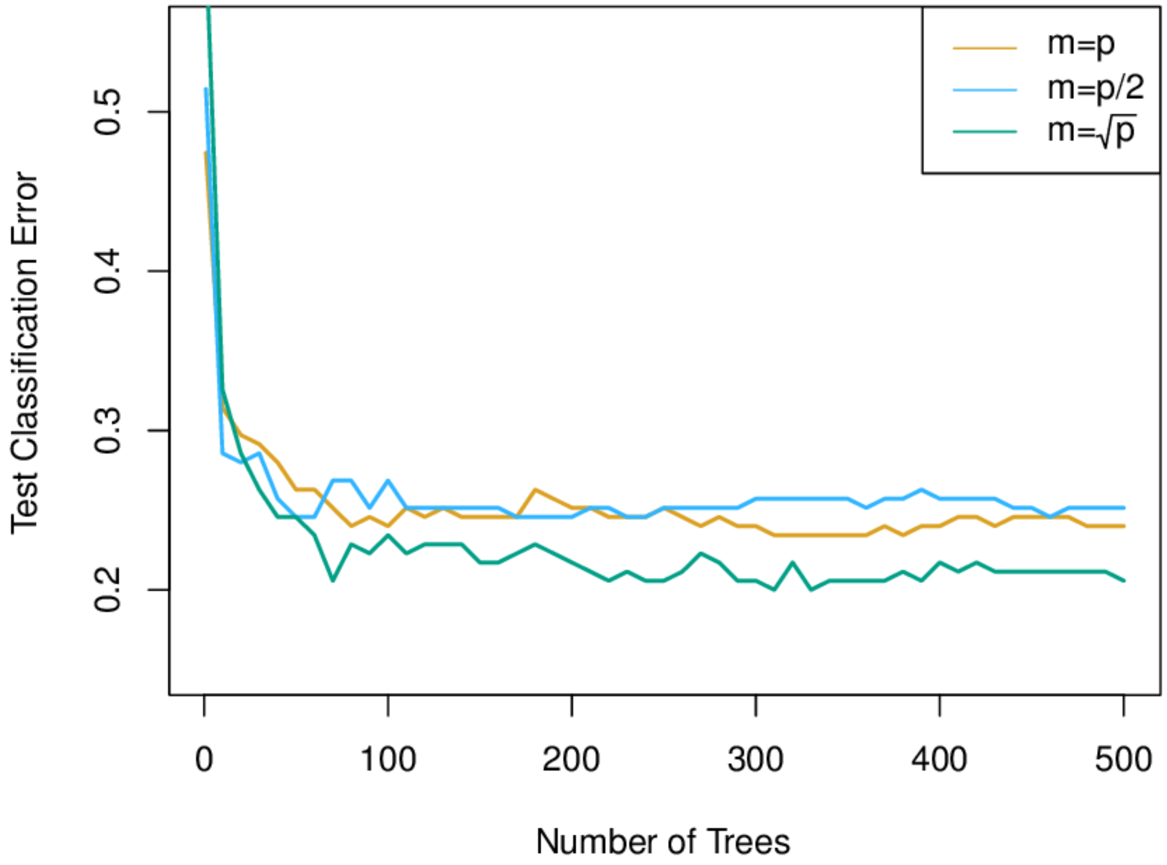
Classification

```
For each of the $t$ decision trees:
   Predict class of instance.
Return class that was predicted most often.
```

Example: Gene Expression Data

- Predict cancer types based on gene expression.
- Outcome: normal or 14 different types of cancer.
- Feature expression measurement of 4718 genes from 349 patients.
  - There are about 20,000 genes in humans, and individual genes have different levels of activity, or expression in particular cells, tissues, and biological conditions.
- Feature selection: select 500 genes that have the largest variance in the training set.
- Randomly divided the observations into a training and a test set.
- Apply random forests to the training set for three different values of the number of splitting variables m.

Result: Gene Expression Data

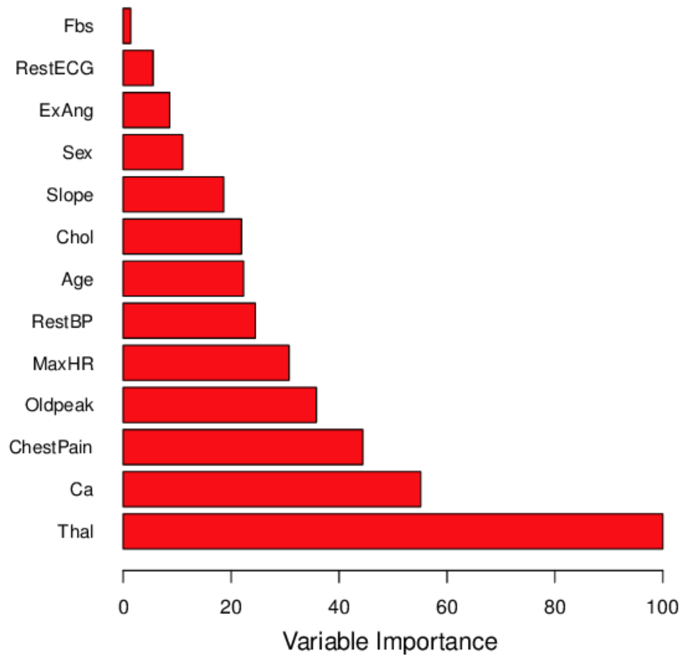


Details of Previous Figure

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Variable Importance Measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Variable importance plot for the `Heart` data

BOOSTING

Boosting

- Also uses voting/averaging but models are 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

Adaboost (Freund and Schapire, 1995)

- Instead of resampling, uses training set re-weighting
  - Each training sample uses a weight to determine the probability of being selected for a training set.
- AdaBoost is an algorithm for constructing a “strong” classifier as a linear combination of “simple” “weak” classifier
- Final classification based on weighted vote of weak classifiers

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Adaboost Terminology

- $h_t(x)$ … “weak” or basis classifier (Classifier = Learner = Hypothesis)
- $H(x) = sign(f(x))$ … “strong” or final classifier

- Weak Classifier: < 50% error over any distribution
- Strong Classifier: thresholded linear combination of weak classifier outputs

Discrete Adaboost Algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$

Initialise $D_1(i) = \frac{1}{m}$.

For $t = 1, \ldots, T$:

- Find the classifier $h_t : X \to \{-1, +1\}$ that minimizes the error with respect to the distribution D_t:

  $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j$, where $\epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
- Prerequisite: $\epsilon_t < 0.5$, otherwise stop.
- Choose $\alpha_t \in \mathbf{R}$, typically $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ where ϵ_t is the weighted error rate of classifier h_t.
- Update:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

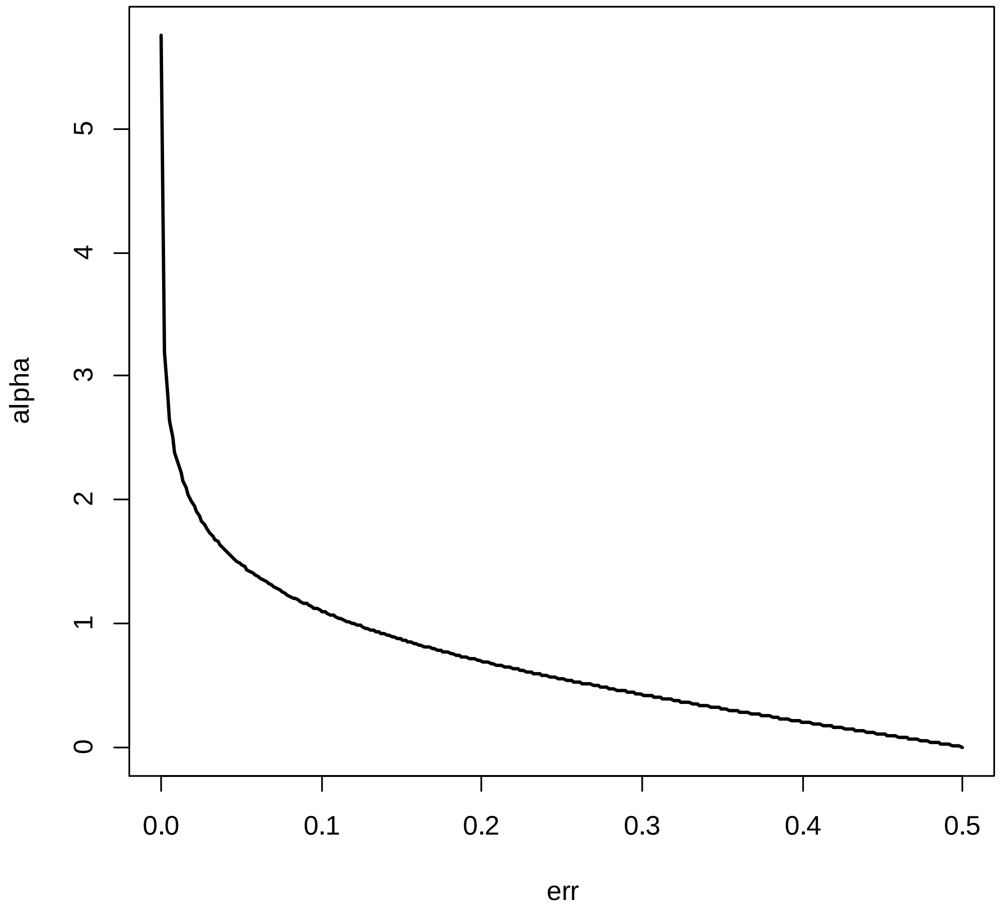
where Z_t is a normalisation factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

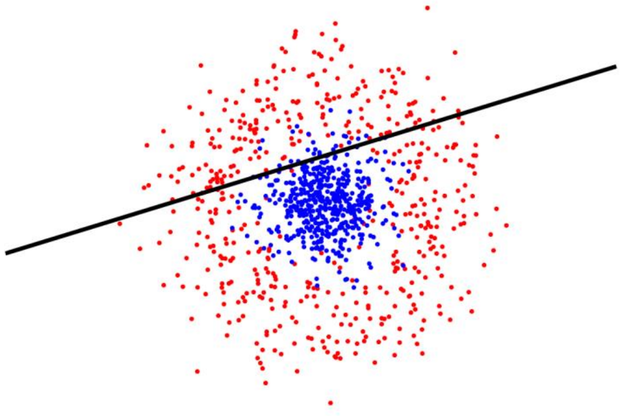
Setting α_t

- α_t is close to 0 if error is close to 0.5.
- α_t will be quite large if error is very small.

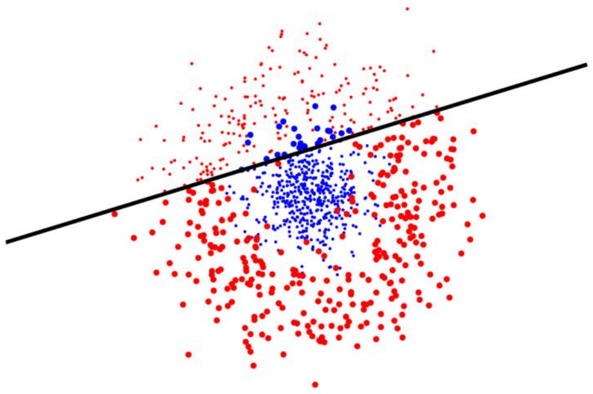


Reweighting

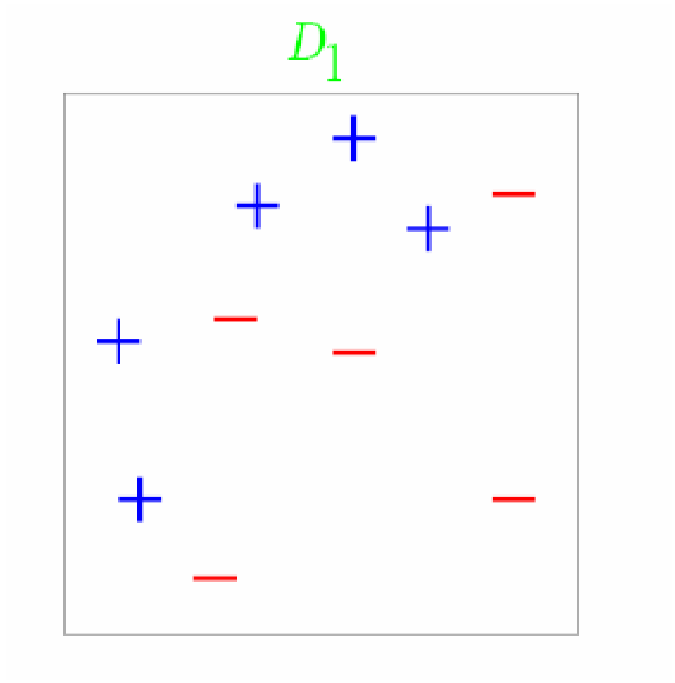
- Effect on the training dataset
- $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
- $\exp(-\alpha_t y_i h_t(i)) \begin{cases} < 1, & if\ y_i = h_t(x_i) \\ > 1, & if\ y_i \neq h_t(x_i) \end{cases}$
- ➔ Increase weight of wrongly classified examples
- ➔ Decrease weight of correctly classified examples.



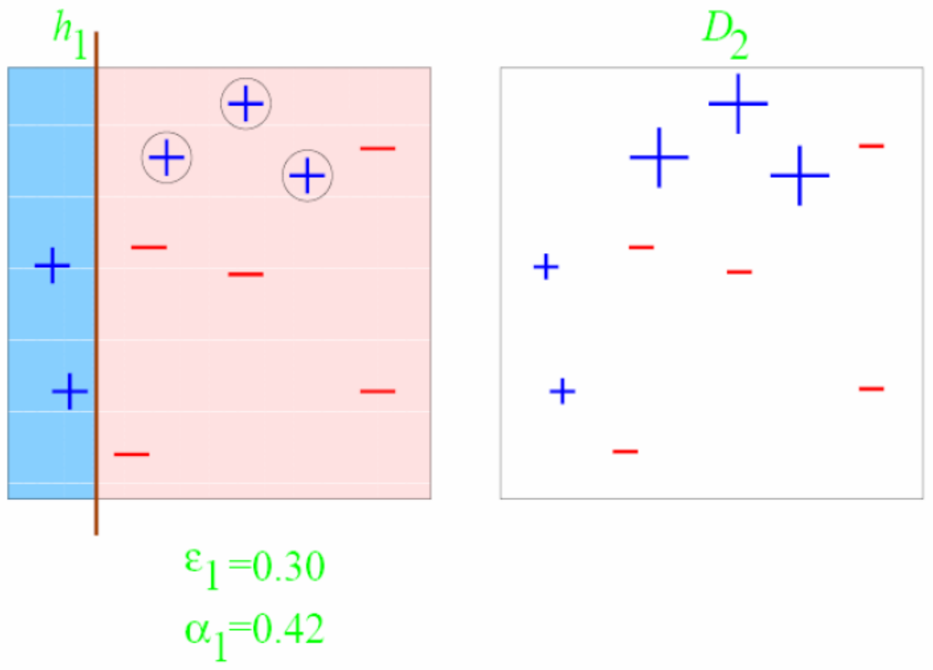
➔



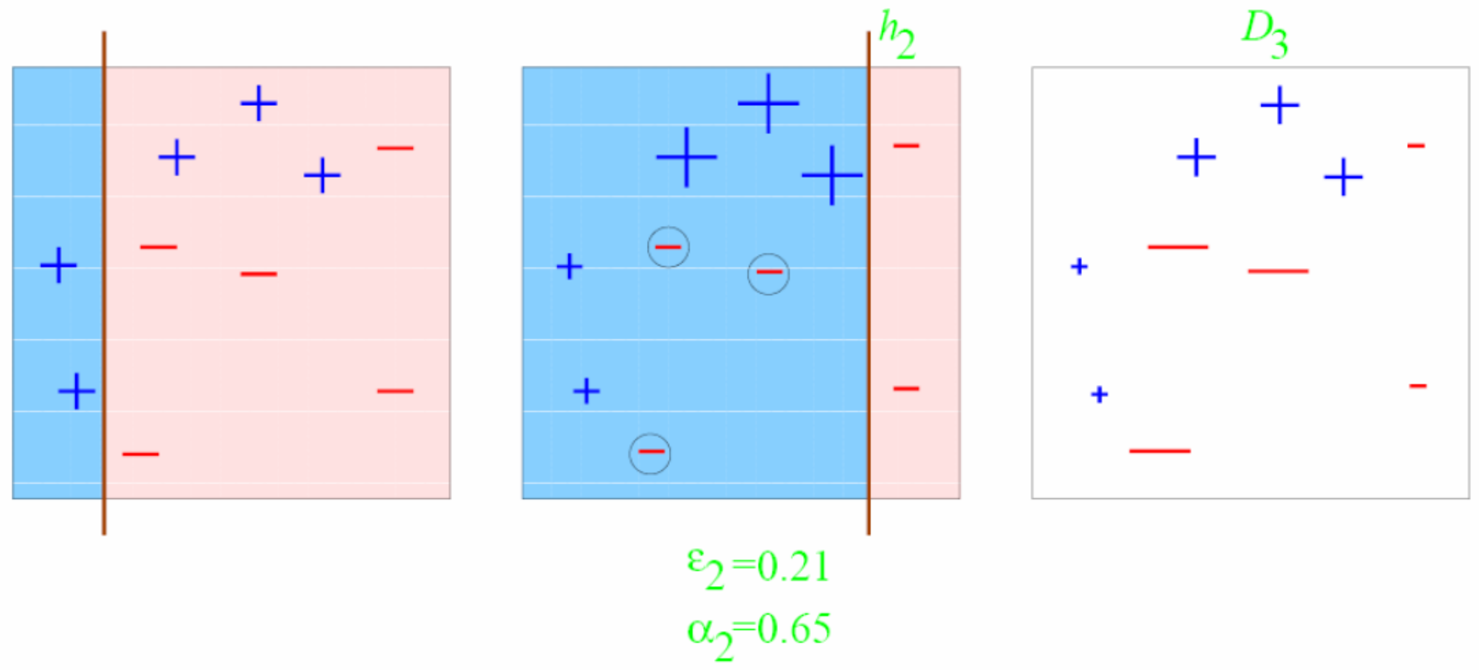
An example: horizontal and vertical half-planes



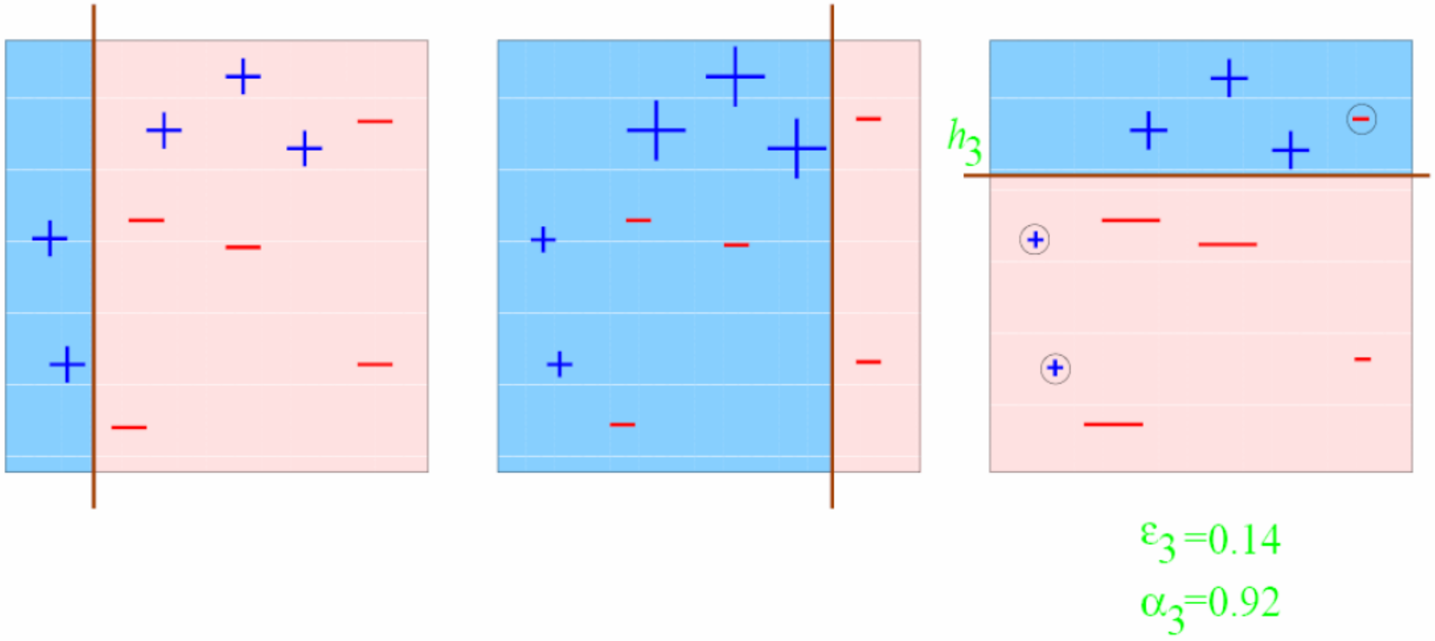
Round 1



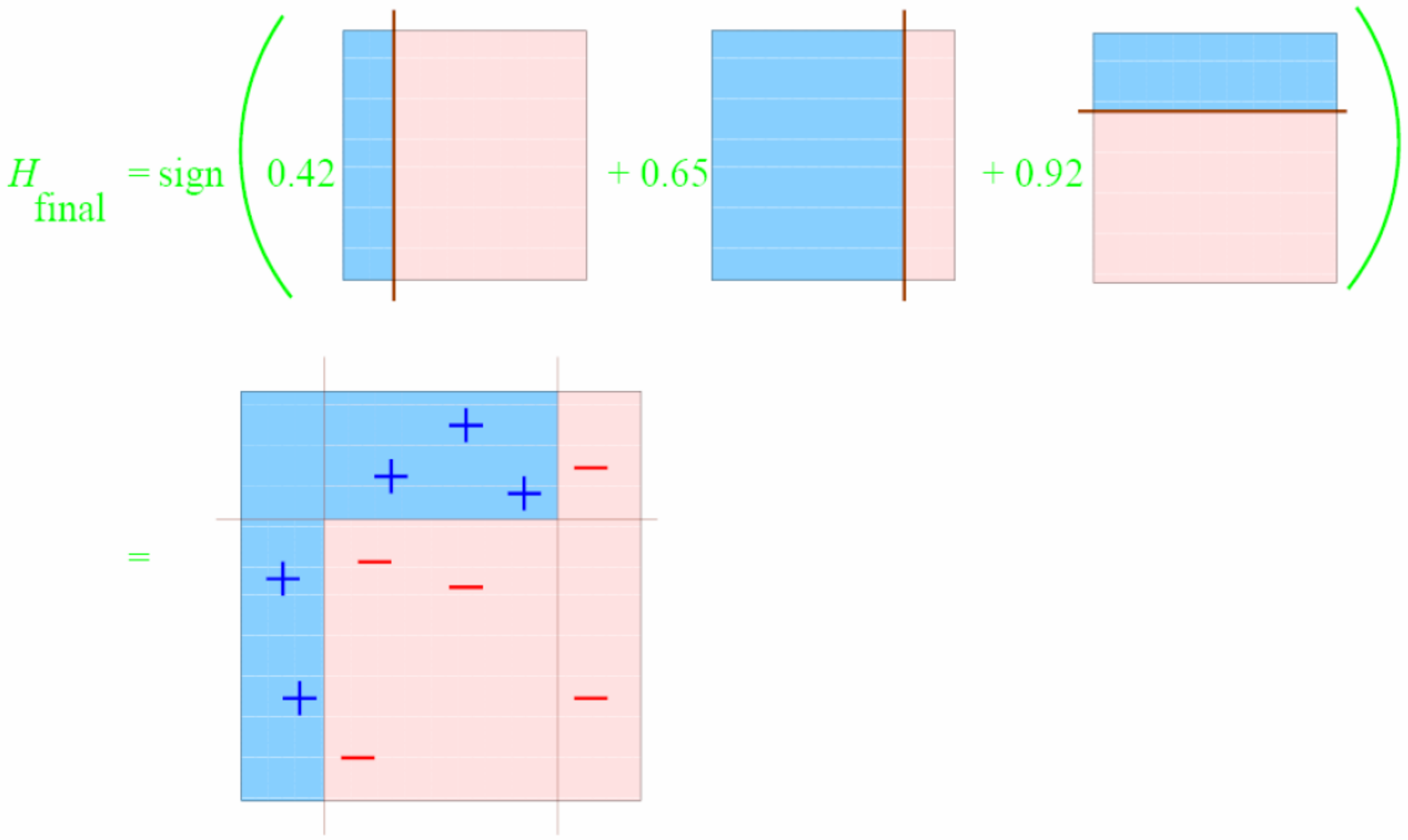
Round 2

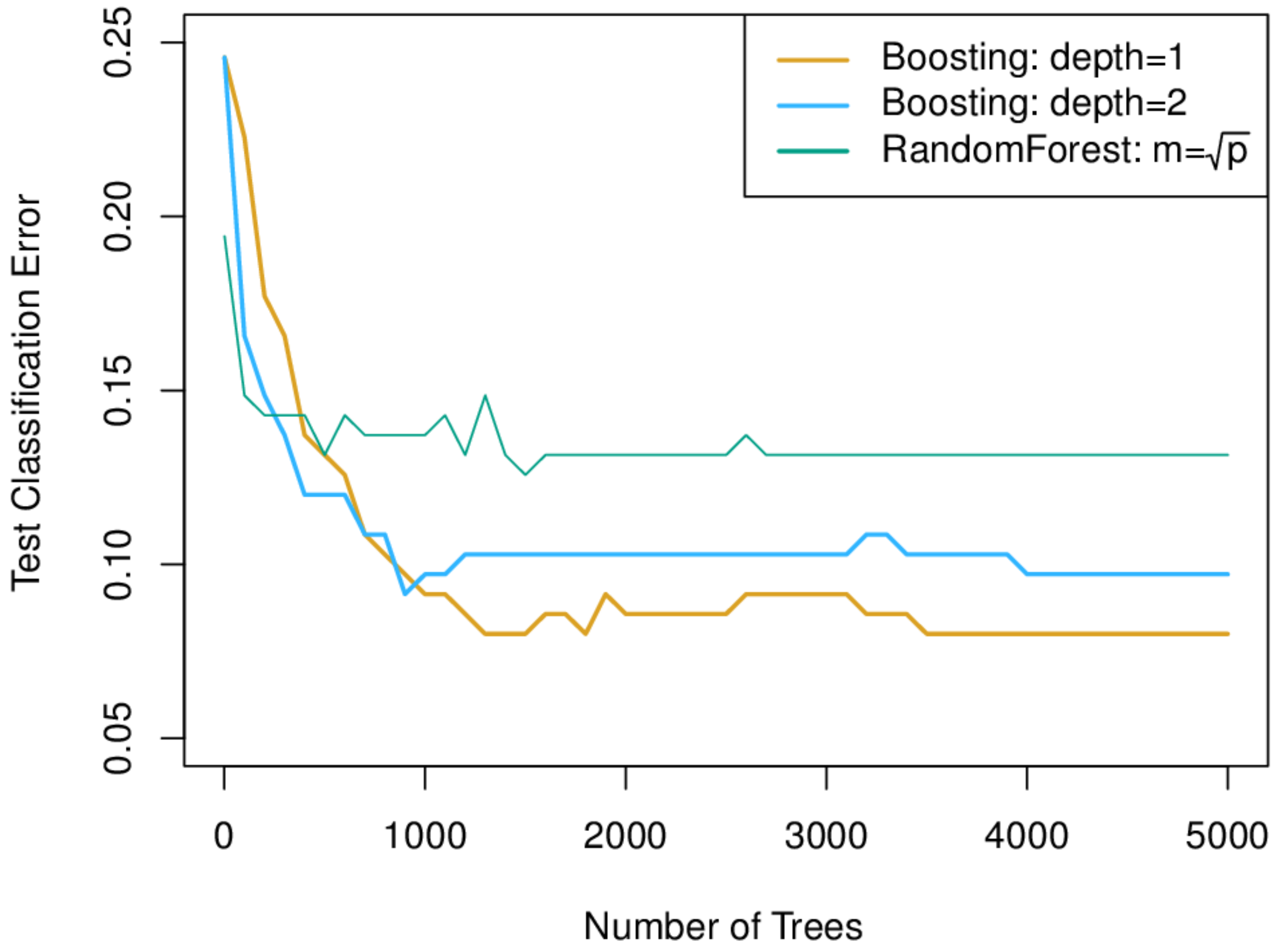


Round 3



Final Classifier




Boosting: depth=1
Boosting: depth=2
RandomForest: m=$\sqrt{p}$
Test Classification Error
0.05
0.10
0.15
0.20
0.25
0
1000
2000
3000
4000
5000
Number of Trees

Details of Previous Figure

- Comparing boosting and random forest.
- Predict cancer versus normal.
- The test error is displayed as a function of the number of trees.
- Two boosted models. Depth-1 trees slightly outperformed depth-2 trees. Both outperformed random forest.
- The test error rate for a single tree is 24%.

Pros and cons of AdaBoost

Advantages

- Very simple to implement
- Usually resulting in relatively simple classifier
- Fairly good generalization

Disadvantages

- Suboptimal solution
- Sensitive to noisy data and outliers

Boosting Algorithm For Regression Tree

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   2.1 Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   2.3 Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

What is the Idea Behind this Procedure?

- Unlike fitting a single large decision tree to the data, which amounts to *fitting the data hard* and potentially overfitting, the boosting approach instead *learns slowly*.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Tuning Parameters for Boosting

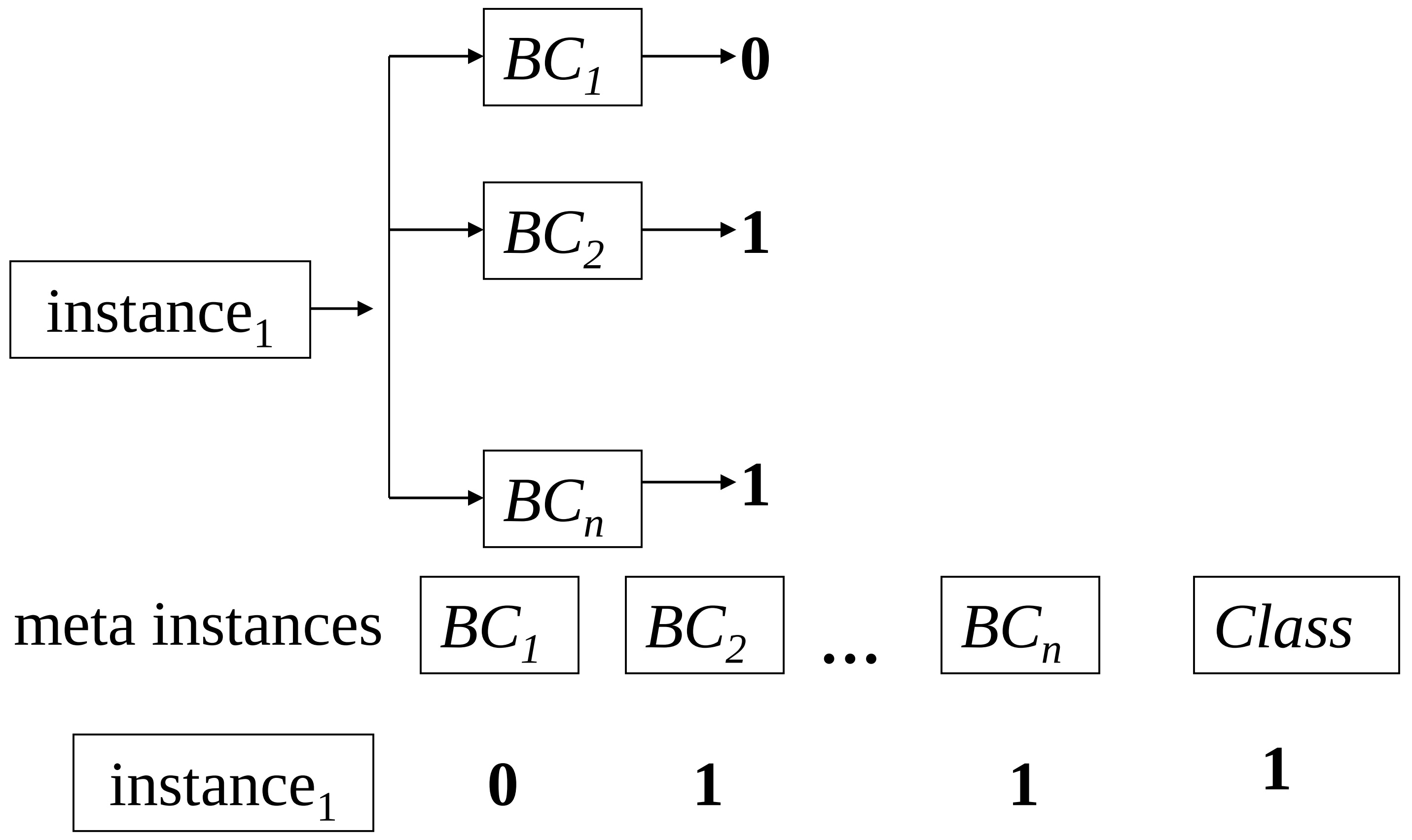
1. The *number of trees* B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The *shrinkage parameter* λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The *number of splits* d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split and resulting in an additive model. More generally d is the *interaction depth*, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

STACKING

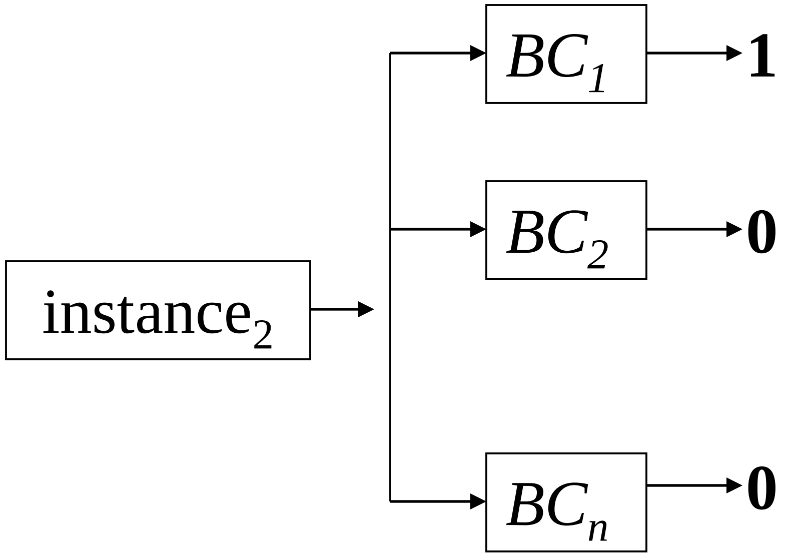
Stacking

- 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)*
- Base learners usually different learning schemes
- Hard to analyze theoretically: “black magic”

Stacking

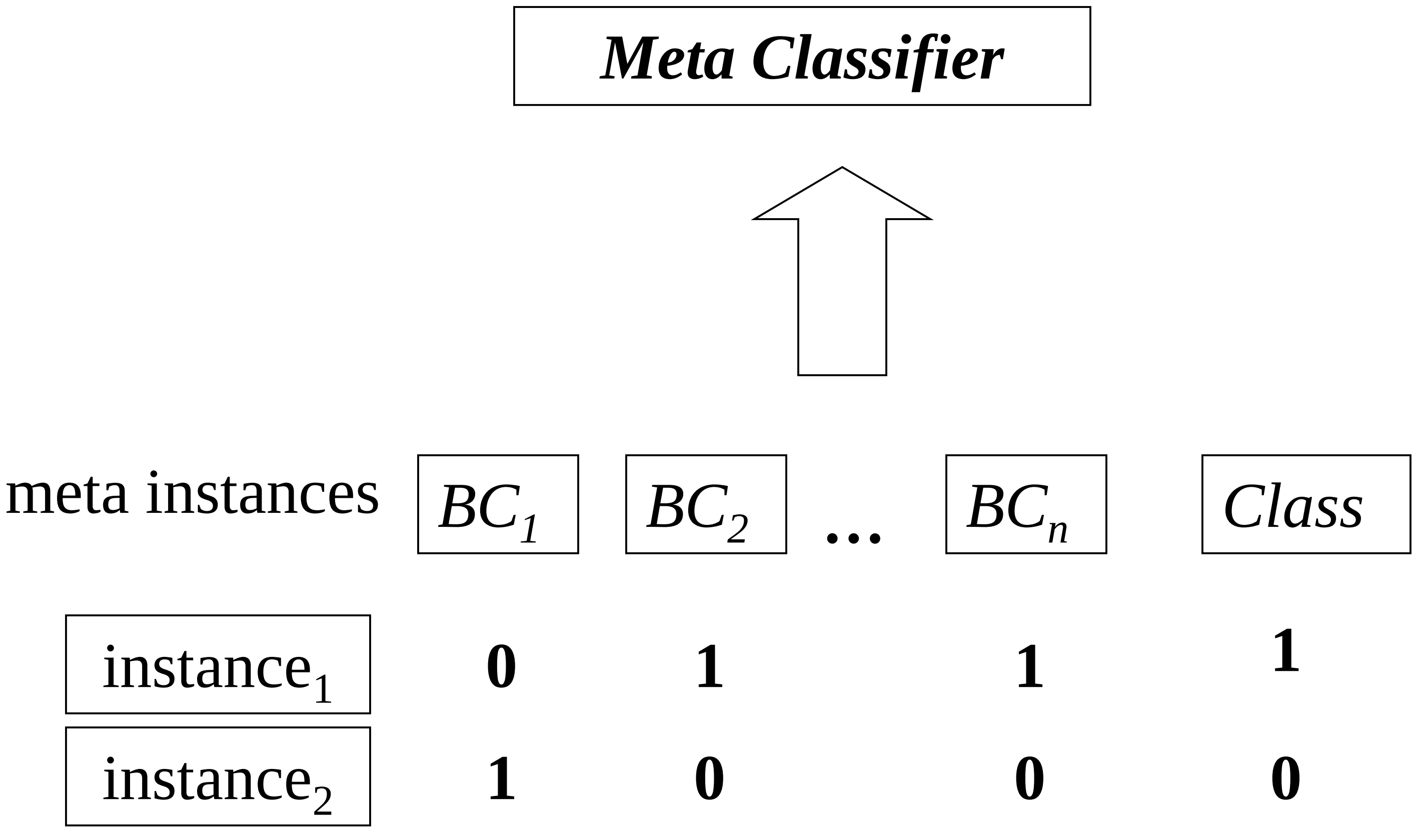

instance$_1$
BC_1
0
BC_2
1
BC_n
1
meta instances
BC_1
BC_2
…
BC_n
Class
instance$_1$
0
1
1
1

Stacking



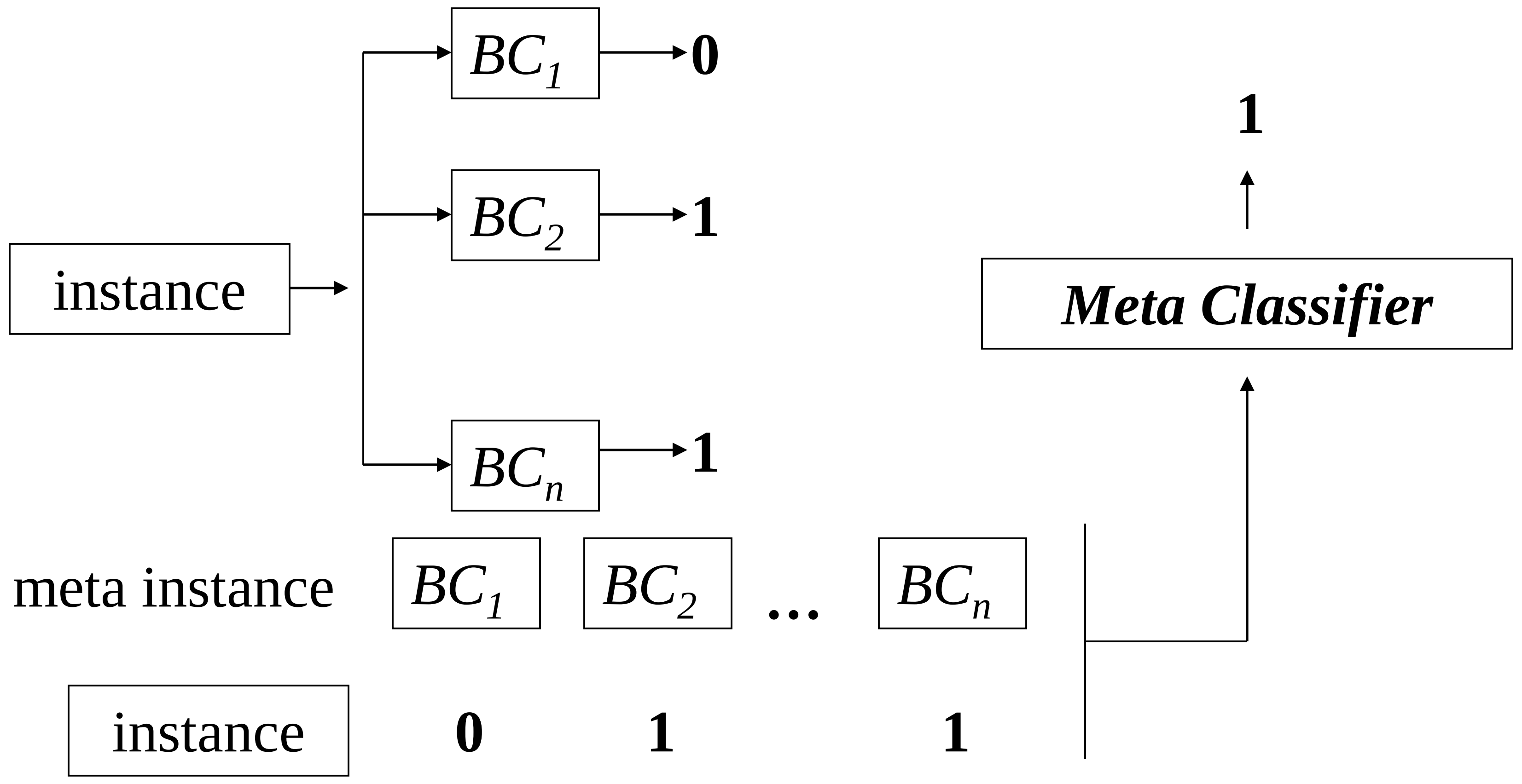
meta instances	BC_1	BC_2	…	BC_n	$Class$
instance_1	**0**	**1**		**1**	**1**
instance_2	**1**	**0**		**0**	**0**

Stacking



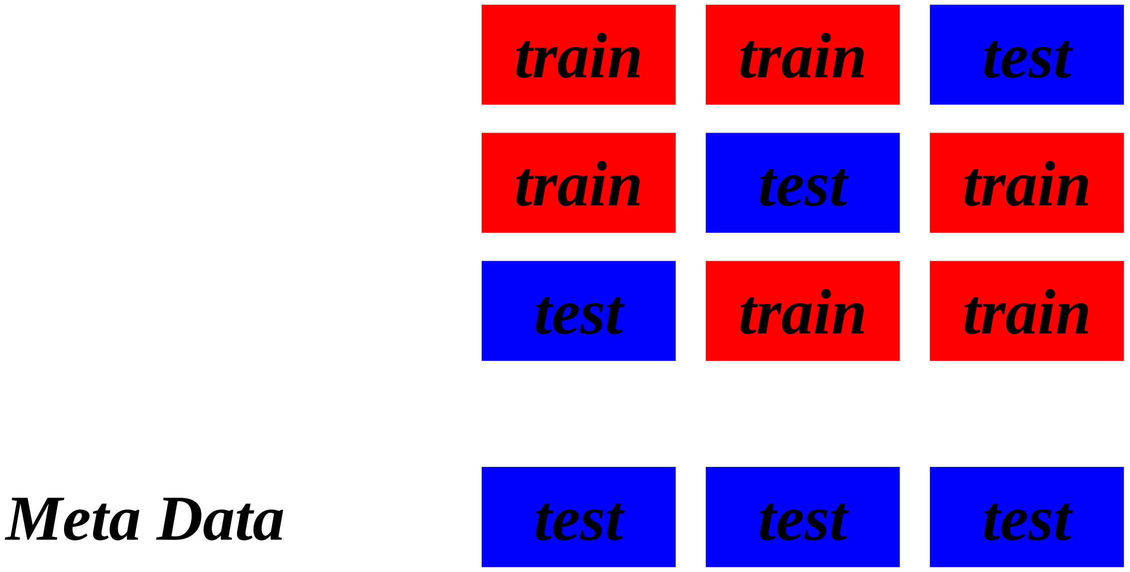
meta instances	BC_1	BC_2	…	BC_n	$Class$
instance_1	**0**	**1**		**1**	**1**
instance_2	**1**	**0**		**0**	**0**

Stacking


instance
BC_1
BC_2
BC_n
0
1
1
1
Meta Classifier
meta instance
BC_1
BC_2
…
BC_n
instance
0
1
1

More on stacking

- Predictions on training data can't be used to generate data for level-1 model! The reason is that the level-0 classifier that better fit training data will be chosen by the level-1 model!
- Thus, k-fold cross-validation-like scheme is employed! An example for k = 3!
- Need an extra "tuning" set for level 1 model.
- Can also train the level-1 model by further splitting the training fold and reserve a "tuning" set for level-1 model.



More on stacking

- If base learners can output probabilities it’s better to use those as input to meta learner
- Which algorithm to use to generate meta learner?
  - In principle, any learning scheme can be applied
  - David Wolpert: “relatively global, smooth” model
    - Base learners do most of the work
    - Reduces risk of overfitting

Summary

- Decision trees are simple and easy-to-interpret models for regression and classification
- However, they are often not as effective as other approaches
- Several methods, such as bagging, random forests, and boosting can be used to improve performance.
- Bagging, boosting are general methods that can applied in combination with other learning approaches
  - If the classifier is unstable (high variance), then apply bagging!
  - If the classifier is stable and simple (high bias) then apply boosting!
- Stacking is another choice when you want to combine results from multiple classifiers/regression models