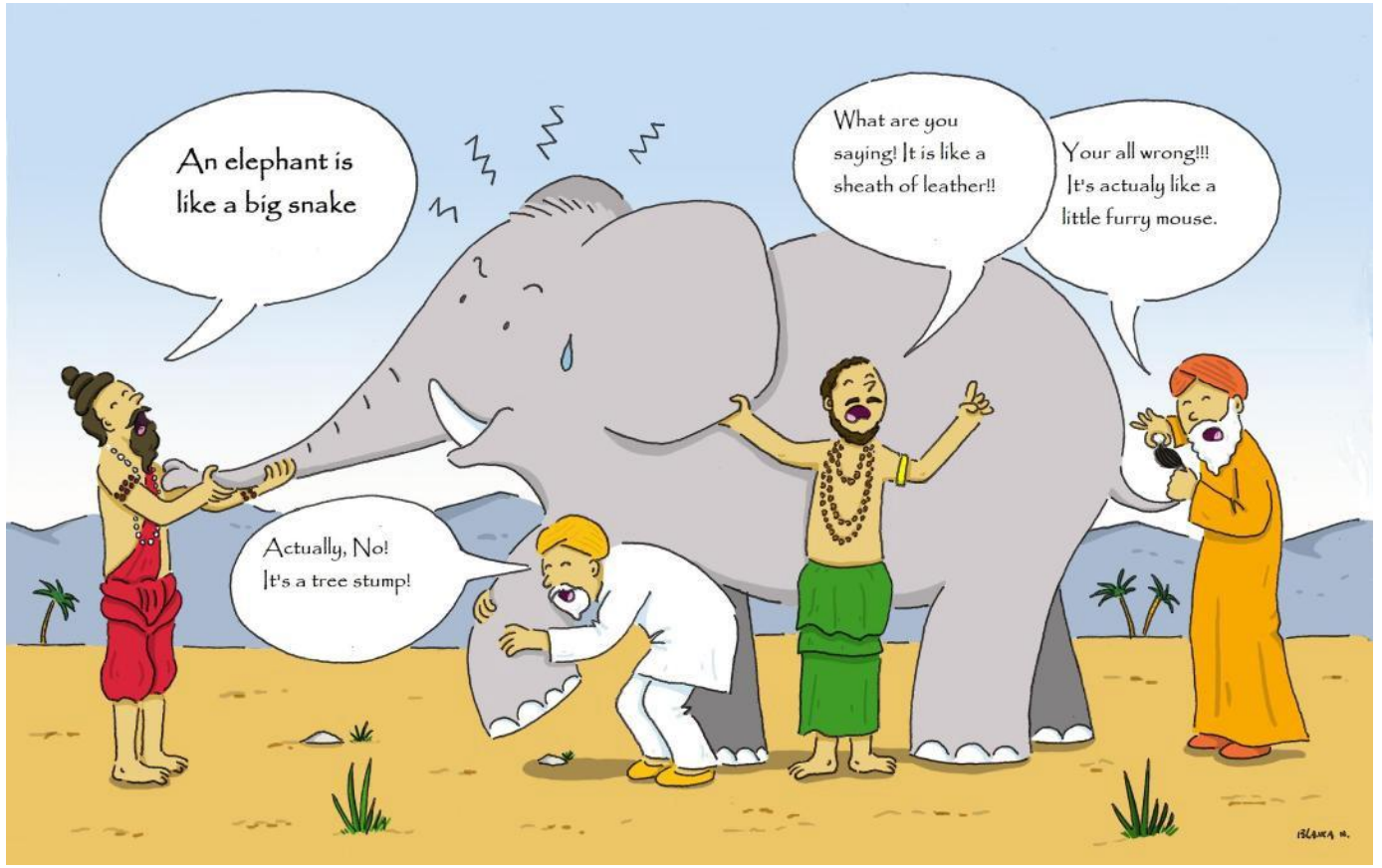
University of Ljubljana, Faculty of Computer and Information Science

Ensemble methods



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, November 2022

Contents

- about ensembles: how & why
- bagging and random forests
- boosting
- stacking
- a few other ideas

How ensembles works?

- learn large number of basic (simple) classifiers
- merge their predictions

- the most successful methods
  - bagging (Breiman, 1996)
  - boosting (Freund & Shapire, 1996)
  - random forest (Breiman, 1999)
  - XGBoost (eXtreme Gradient Boosting) (Chen & Guestrin, 2016)

Why ensembles work?

- we need different classifiers
  - different in a sense that they produce correct predictions on different instances
- the law of large numbers does the rest
- guidelines for basic classifiers
  - different
  - as strong as possible, but at least weak
- a weak classifier is an expression from computational learning theory (COLT), it means a classifier whose performance is at least $\in > 0$ better than a random classifier

Bagging and random forests

- Bagging
  - sample selection with bootstrapping
  - Bagging for regression trees
  - Bagging for classification trees
  - Out-of-bag error estimation
  - Variable importance: relative influence plots

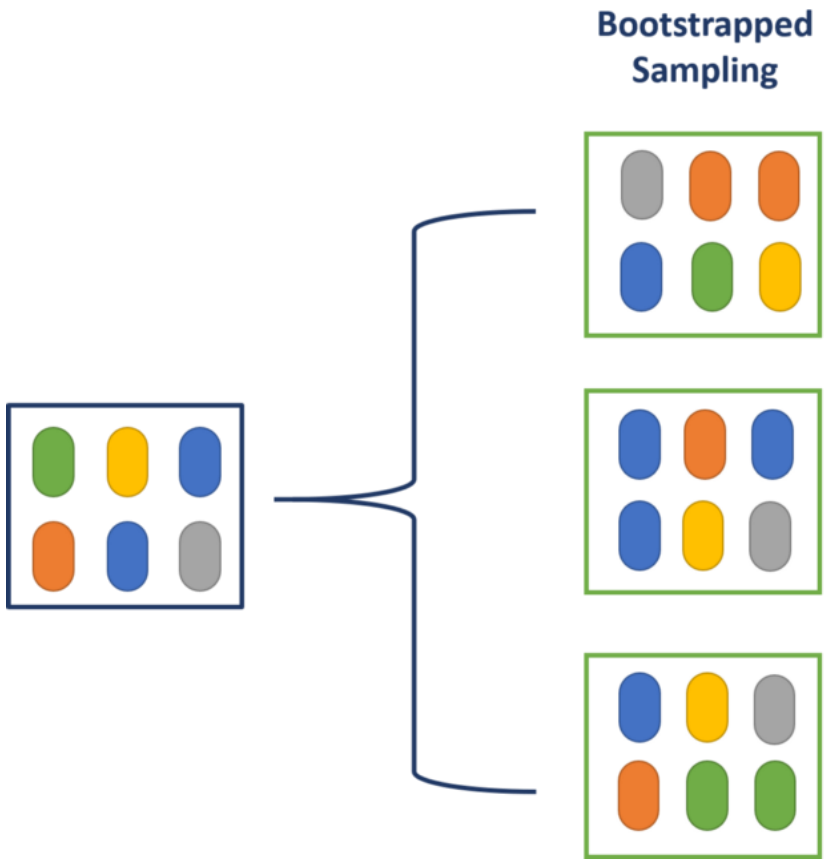
- Random Forests

Bagging

- Decision trees suffer from high variance!
  - If we randomly split the training data into 2 parts, and fit decision trees on both parts, the results of different runs could be quite different
- We would like to have models with low variance
- To solve this problem, we can use bagging (***b***ootstrap ***agg***regat***ing***).

Bootstrapping

- Resampling of the observed dataset (and of equal size to the observed dataset), each of which is obtained by random sampling with replacement from the original dataset.



Bootstrapping

- Draw instances from a dataset *with replacement*
- Probability that we do not pick an instance after N draws

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368$$

that is, only 63.2% of instances are used in one draw

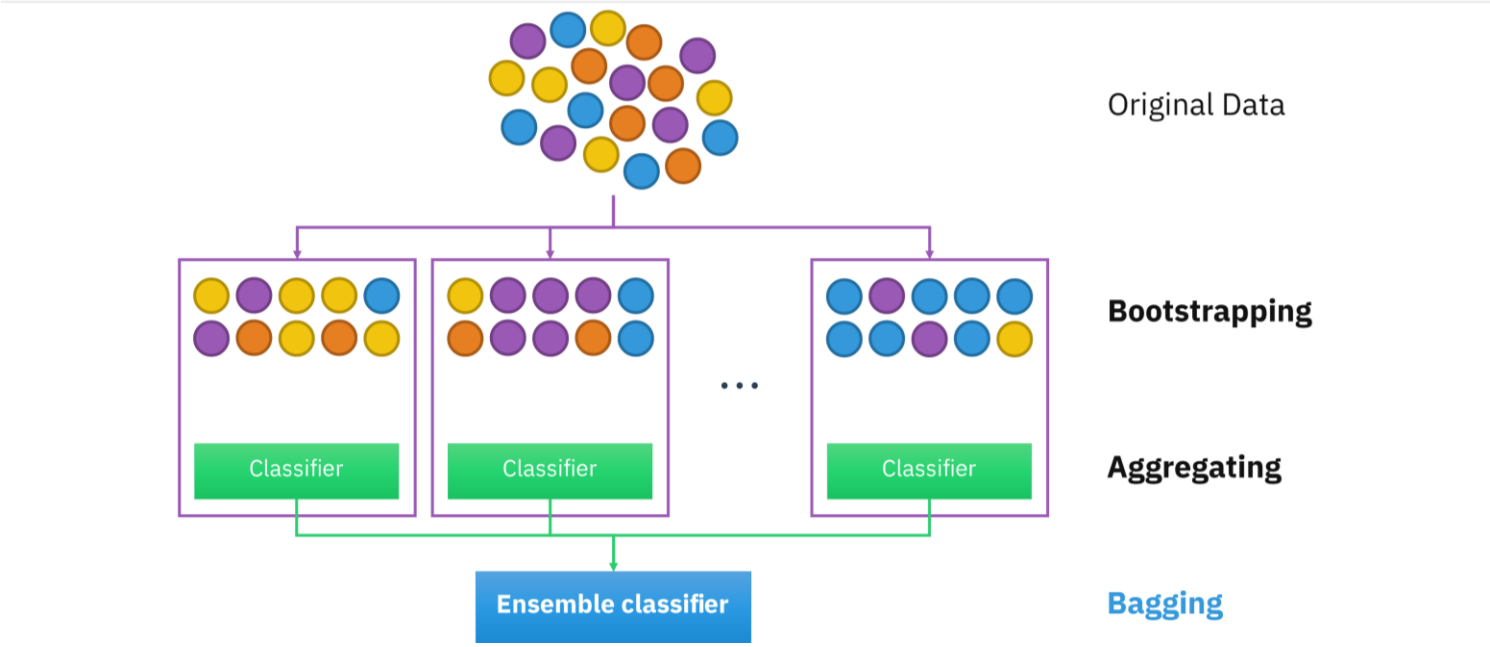
What is bagging?

- Bagging is a powerful idea based on two things:
  - Averaging: reduces variance!
  - Bootstrapping: plenty of training datasets!
- Why does averaging reduces variance?
  - Averaging a set of observations reduces variance.
  - Given a set of *n* independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by σ^2/n.

How does bagging work?

- Generate B different bootstrapped training datasets

- Train the statistical learning method on each of the B training datasets, and obtain the prediction



Bagging for regression trees

- Construct B regression trees using B bootstrapped training datasets
- Average the resulting predictions

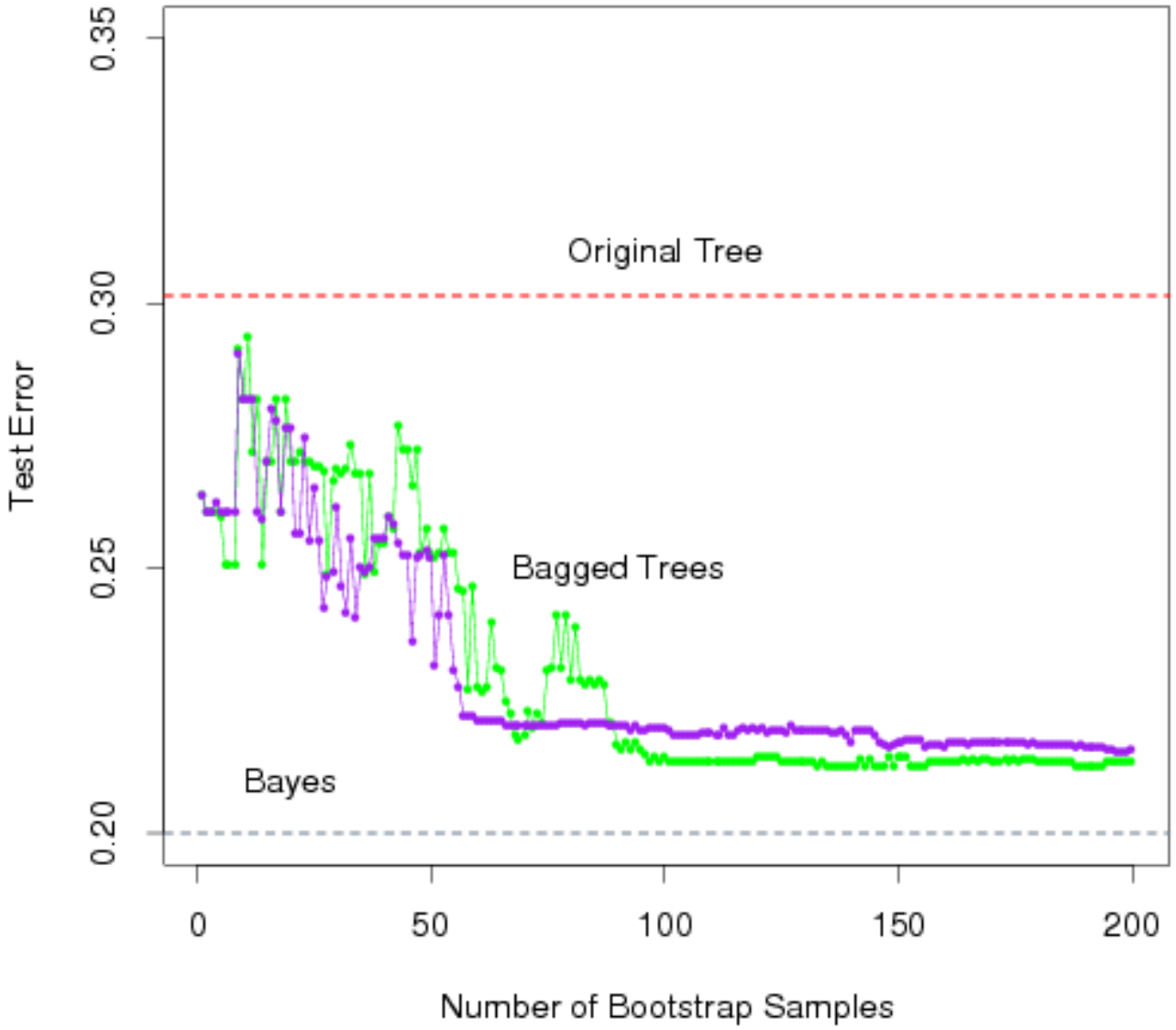
- The trees are not pruned, so each individual tree has high variance but low bias.
- Averaging these trees reduces variance, and thus we end up lowering both variance and bias ☺

Bagging for classification trees

- Construct B decision trees using B bootstrapped training datasets
- For 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.

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).



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 1- 1/e ≈ 63% of the observations, so we end up having 1/e ≈ 37% of the observations useful for testing

Variable importance measure

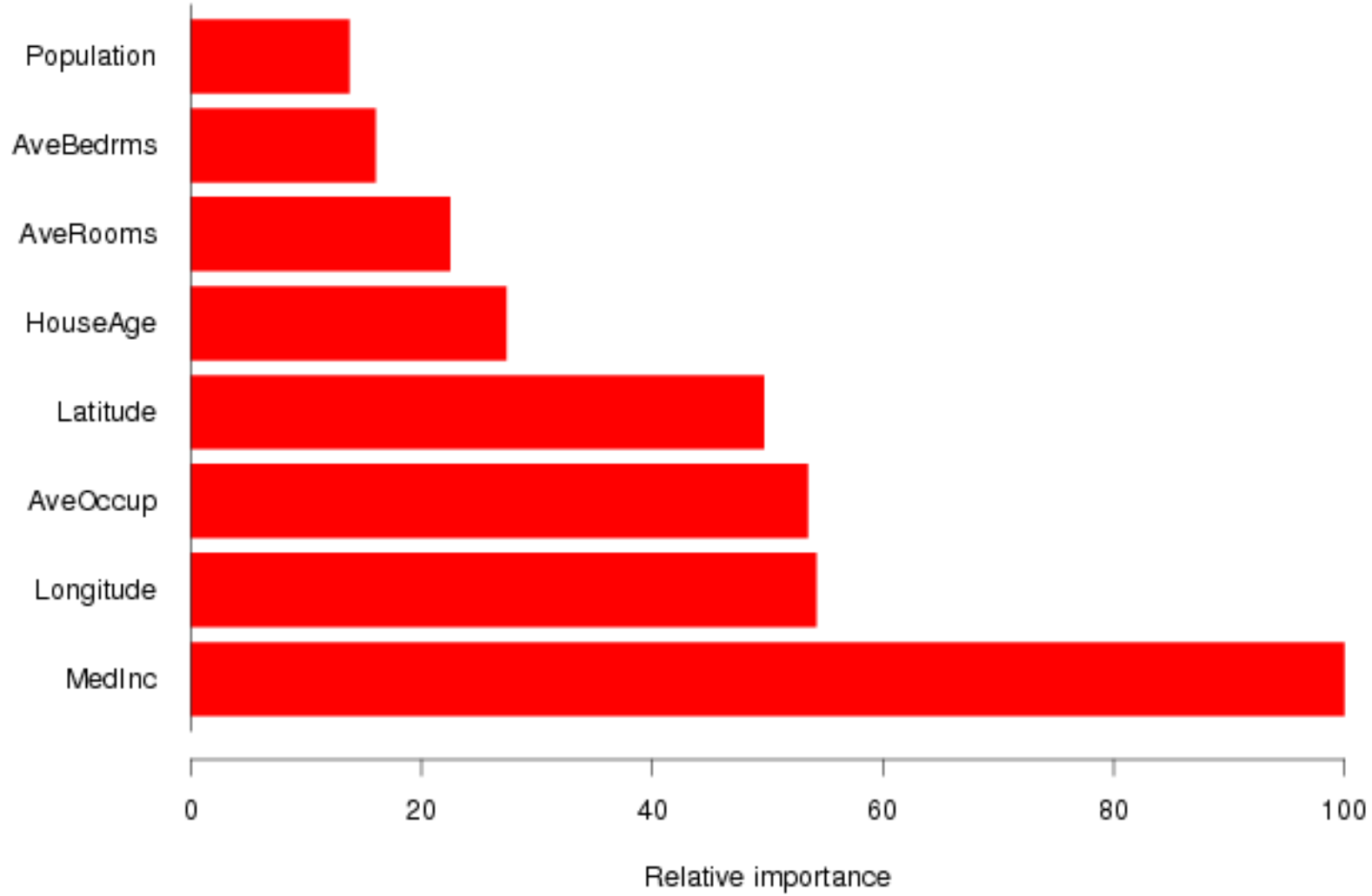
- Bagging typically improves the accuracy over prediction using a single tree, but it is now hard to interpret the model!
- We have hundreds of trees, and it is no longer clear which variables are most important to the procedure
- Thus bagging improves prediction accuracy at the expense of interpretability
- But, we can still get an overall summary of the importance of each predictor using relative influence plots

Relative influence plots

- How do we decide which variables are most useful in predicting the response?
  - We can compute something called relative influence plots.
  - These plots give a score for each variable.
  - These scores represents the decrease in MSE when splitting on a particular variable
  - A number close to zero indicates the variable is not important and could be dropped.
  - The larger the score the more influence the variable has.

Example: Housing data

- Median Income is by far the most important variable.
- Longitude, Latitude and Average occupancy are the next most important.



Random forests

- It is a very efficient statistical learning method
- It builds on the idea of bagging, but it provides an improvement because it de-correlates the trees
- How does it work?
  - Build a number of decision trees on bootstrapped training sample,
  - When building these trees, each time a split in a tree is considered, a random sample of *m* predictors is chosen as split candidates from the full set of *p* predictors.
  - Usually $m \approx \sqrt{p}$ or $m \approx 1 + \log_2 p$

Why are we considering a random sample of *m* predictors instead of all *p* predictors for splitting?

- Suppose that we have a very strong predictor in the data set along with a number of other moderately strong predictors, then in the collection of bagged trees, most or all of them will use the very strong predictor for the first split!
- All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated
- Averaging many highly correlated quantities does not lead to a large variance reduction, and thus random forests “de-correlates” the bagged trees leading to more reduction in variance

Properties

- low classification (and regression) error
- no overfitting
- robust concerning the noise and the number of attributes
- relatively fast
- learning instances not selected with bootstrap replication are used for evaluation of the tree (oob = out-of-bag evaluation)

Out-of-bag evaluation

- on average 1/e ~ 37% of the learning set is not used to train each of the basic classifiers
- classification margin

$$mr(\mathbf{x}, y) = P(h(\mathbf{x}) = y) - \max_{\substack{j=1 \\ j \neq y}}^{c} P(h(\mathbf{x}) = j)$$

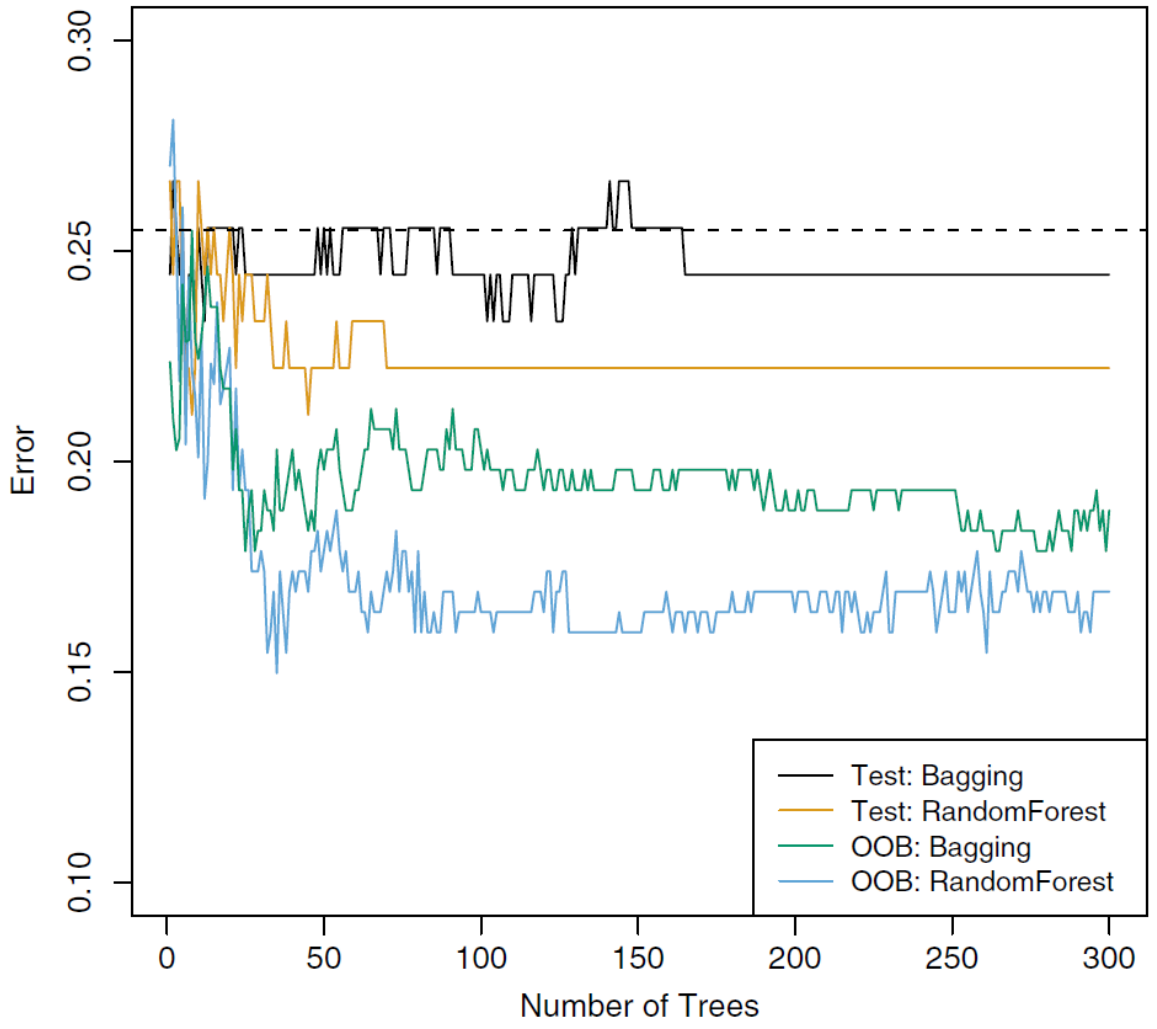
  - mr is estimated with all classifiers where **x** is in oob set
- strength of the forest = average margin over training or OOB set
- correlation of the trees in forest

$$\bar{\rho} = \frac{\mathrm{var}(mr)}{std(h())^2}$$

- we want high strength and low correlation

OOB-error estimate

- with large number of trees, the OOB estimate is roughly equivalent to the CV error estimate
- computationally much cheaper than CV
- still overly optimistic



Heart data set

RF attribute evaluation

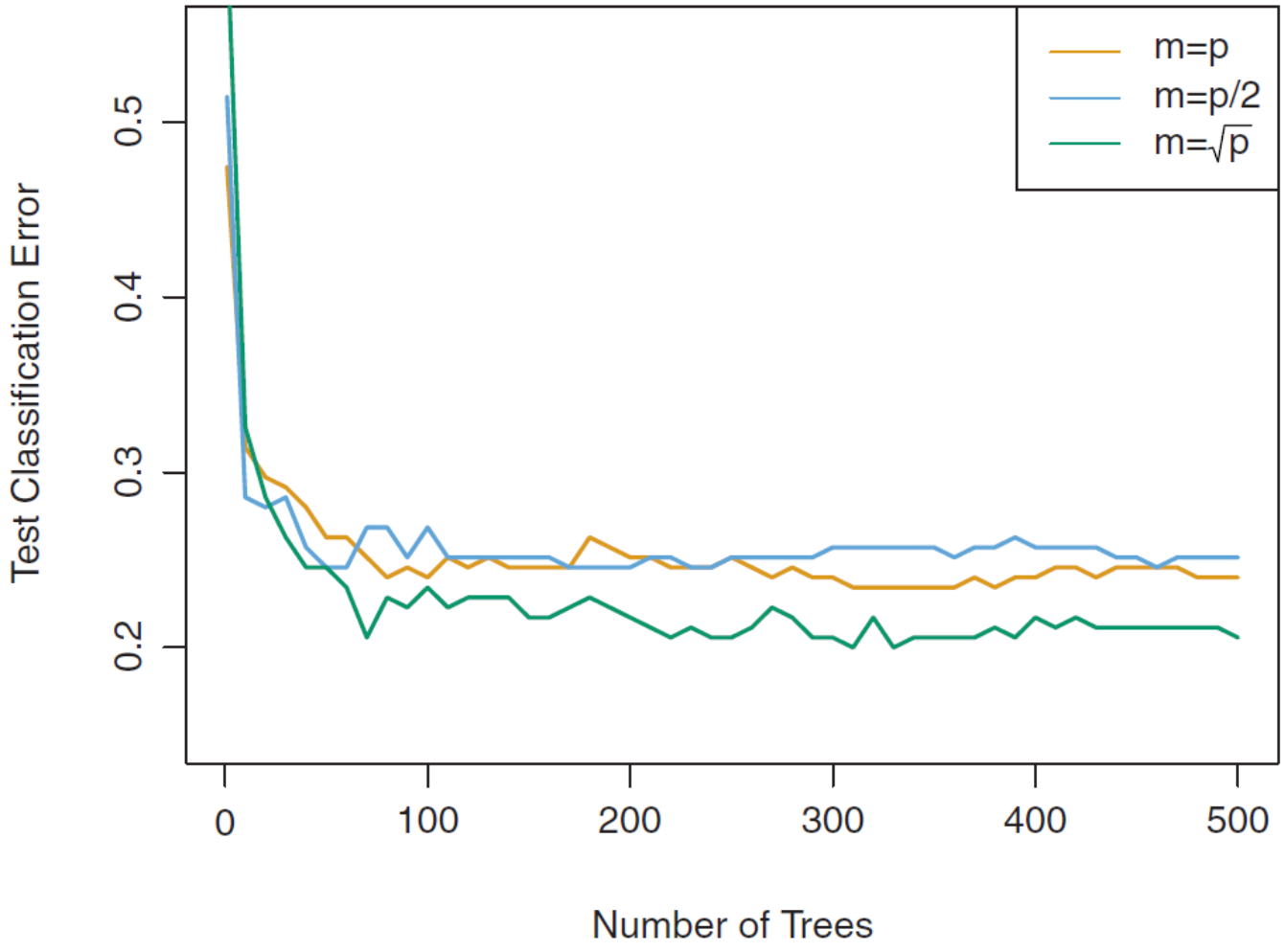
- evaluation of attribute A is the difference between
  - strength of the forest and
  - strength of the forest when values of A are randomly shuffled
- evaluated on the OOB set
- detects also strong conditional dependencies
- works also on an instance-level like nomogram (evaluates only the trees where the instance is in the OOB set)

Similarity of instances

- build instance similarity matrix
- when two instances end in the same leaf of the tree we increase their similarity score
- average over all trees gives similarity measure
- we use that similarity measure to:
  - detect outliers
  - determine typical cases for each class
  - scaling
  - missing values
  - clustering
  - visualization

Random forest with different values of "*m*"

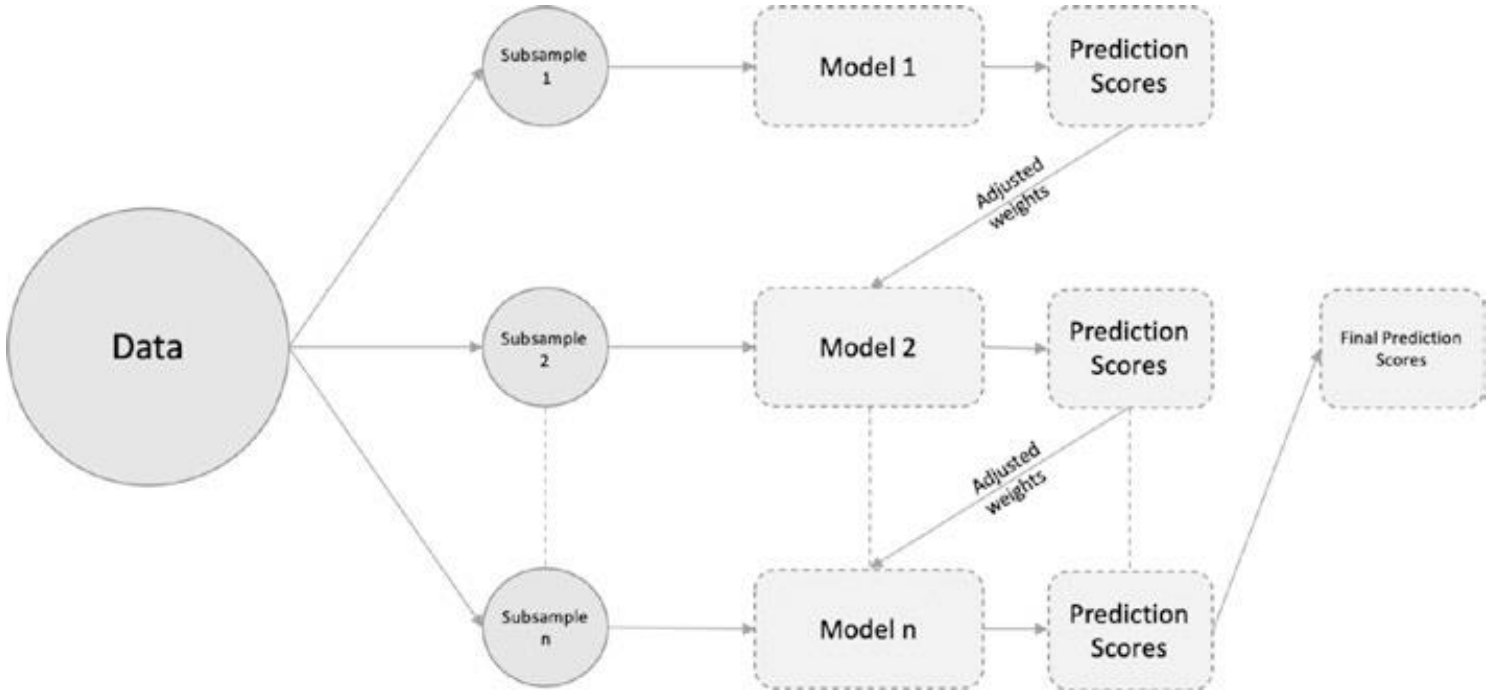
- Notice: when random forests are built using $m = p$, then this amounts to bagging.



Gene expression data (15 classes)

Boosting

- another ensemble method
- grows trees sequentially: each added tree uses information about errors of previous trees



Pseudocode for boosting in regression

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:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

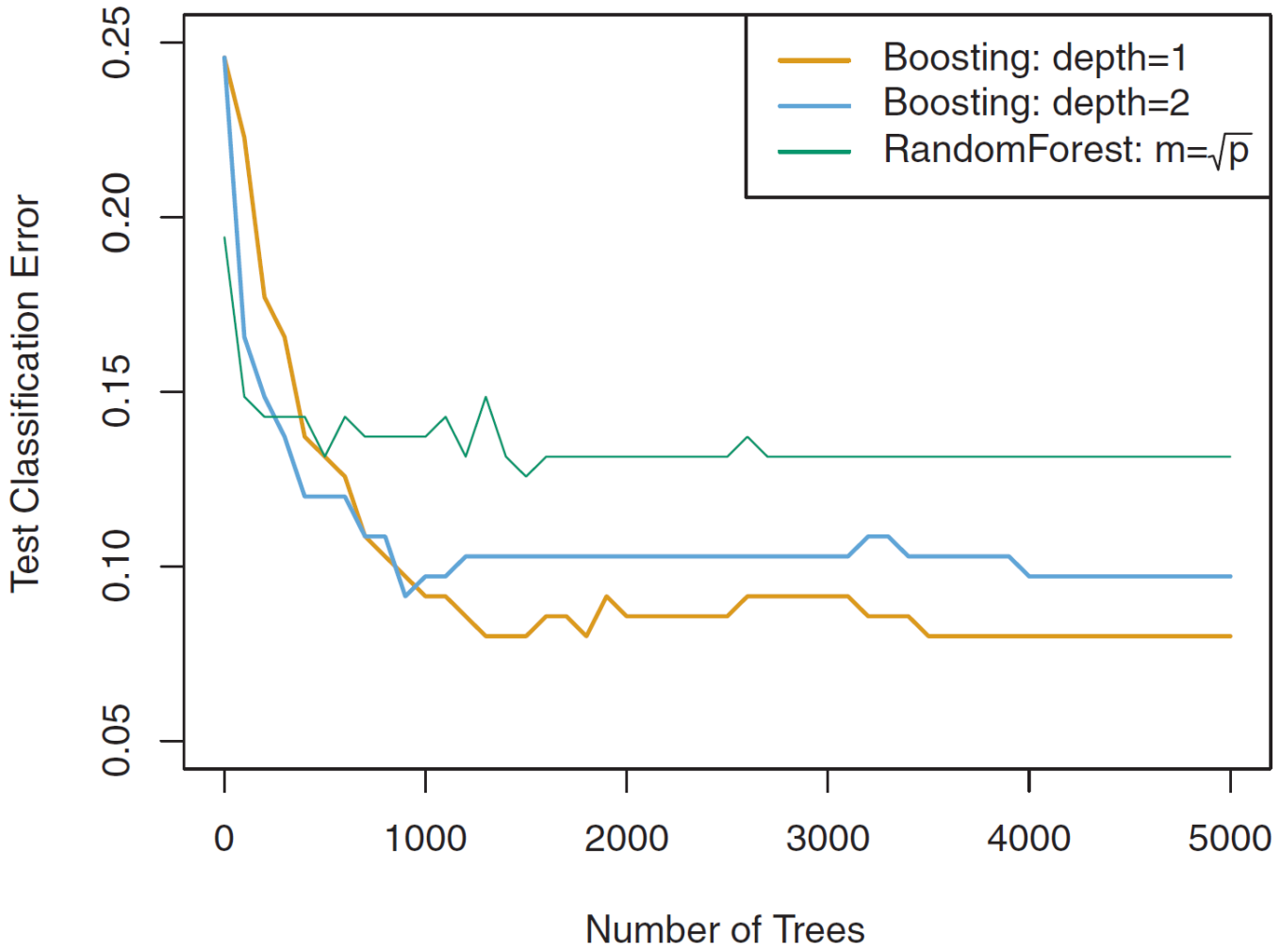
3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Boosting

- each tree takes into account residuals (i.e. errors) of previous trees
- each tree is small, containing only d splits (e.g., d=1, decision stumps)
- learning is slow, controlled by λ
- Parameters of boosting in regression
  - The number of trees *B*, selected with CV, boosting can overfit.
  - The shrinkage parameter *λ*, a small positive number (e.g., 0.01 or 0.001), problem dependent; small *λ* requires large B to achieve good performance
  - The number *d* of splits in each tree, which controls the complexity of the boosted ensemble. Often *d* = 1 works well, but *d* also controls interaction order (*d* splits can contain at most *d* variables).

Boosting performance



Gene expression data (15 classes)
error of single tree is approx. 0.24, std. error around 0.02

Boosting in classification

- AdaBoost, Freund & Shapire, ICML, 1996
  - training instances are weighted according to the success of their classification in the previous iteration
    - increase weight of misclassified instances
    - decrease weight of correctly classified instances
    - the learning focus is transferred to the most difficult instances
  - final classification is a weighted voting of basic classifiers
- deterministic algorithm, works because training sets are different
- mostly better than bagging
- this original version can suffer from overfitting but there are better variants

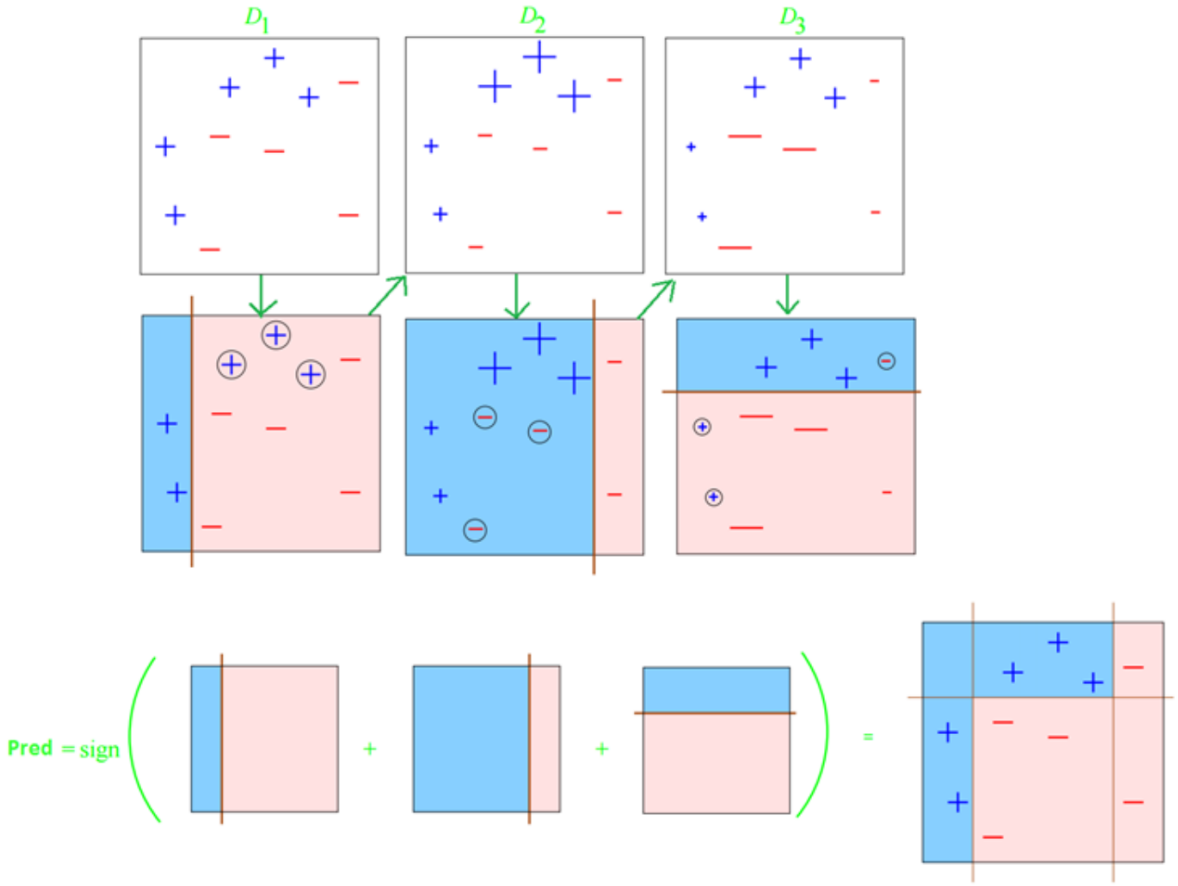
AdaBoost (Freund and Schapire, 1996)

- Given a set of *d* class-labeled instances, $(\boldsymbol{X_1}, y_1), \ldots, (\boldsymbol{X_n}, y_n)$
- Initially, all the weights of instances are set the same (*1/n*)
- Generate *k* classifiers in *k* rounds. At round *i*,
  - Instances from D are sampled (with replacement) or reweighted to form a training set D_i of the same size
  - Each instance's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If an instance is misclassified, its weight is increased, otherwise it is decreased
- Error rate: $err(\boldsymbol{X_j})$ is the misclassification error of instance $\boldsymbol{X_j}$.
  Classifier M_i error rate is the sum of the weights of the misclassified instances:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- The weight of classifier M_i's vote is $\log \frac{1-error(M_i)}{error(M_i)}$

AdaBoost Example



XGBoost – eXtreme Gradient Boosting

Additive model with loss L:

$$\min_{\alpha_{n=1:N},\beta_{n=1:N}} L\left(y, \sum_{n=1}^{N} \alpha_n f(x, \beta_n)\right)$$

GB approximately solves this objective iteratively and greedily:

$$\min_{\alpha_n,\beta_n} L\left(y, f_{n-1}((x) + \alpha_n f_n(x, \beta_n)\right)$$

Chen & Guestrin(2016), XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* https://arxiv.org/abs/1603.02754

https://xgboost.readthedocs.io/en/latest/build.html#r-package-installation

Other possibilities for tree ensembles

- sampling in RF:
  - *p*-sampling without replacement (sampling the proportion of *p* instances, e.g., *p*=10%)
- limiting the size of the trees in RF and bagging
  - more trees needed
- reduced computational complexity
- regularization

Weighting of the trees

- not all trees are equally important (absolutely and in all parts of an instance space)
- weight the trees according to the data
- assume linear combination of base coefficients

$$F(x, a) = a_0 + \sum_{j=1}^{T} a_j \, t_j(x)$$

- solve for coefficients *a*

Penalization

$$\hat{\mathbf{a}} = \arg\min_{a} \frac{1}{N} \sum_{i=1}^{n} L\left(y_{i,} a_0 + \sum_{j=1}^{T} a_j \, t_j(x_i)\right)$$

- direct minimization gives poor generalization, therefore penalize

$$\hat{\mathbf{a}}(\lambda) = \arg\min_{a} \left(\frac{1}{N} \sum_{i=1}^{n} L\left(y_{i,} a_0 + \sum_{j=1}^{T} a_j \, t_j(x_i)\right) + \lambda P(\mathbf{a}) \right)$$

Common penalty functions

- ridge regression

$$P_2(\mathbf{a}) = \sum_{j=1}^{T} |a_j|^2$$

- lasso, sure-shrink

$$P_1(\mathbf{a}) = \sum_{j=1}^{T} |a_j|$$

- solve with gradient descent algorithms (Friedman & Popescu, 2003)

Local weighting

- regularization: global importance of base models
- local importance: local regularization, weighting with margin of similar instances

Locally weighted voting for RF

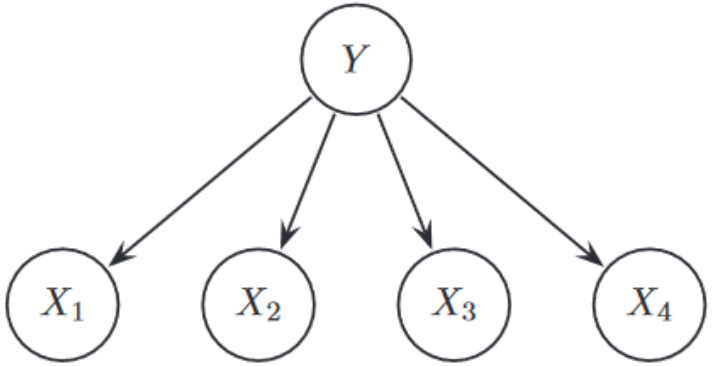
- observation: not all trees are equally good in all parts of the problem space
- opportunity: use OOB instances to locally evaluate the quality of trees
- locality: forest defines the similarity between instances

Weighted voting algorithm for RF

- in classification of a new instance
  - find *t* most similar instances
  - classify each of the similar instances with the trees where it is in the OOB set, and record the margin for the trees
  - compute weights of the trees as the average recorded margin (for trees with negative margin set the weight to zero)
  - forest classification is the weighted voting of the trees

Marko Robnik-Šikonja: Improving random forests. In *Proceedings of European conference on machine learning ECML'04*, pp. 359-370. 2004.

Naïve Bayes based ensembles



- Naive Bayes is a probabilistic classifier

$$\begin{aligned} \underset{y}{\operatorname{argmax}}\, P(y \mid \mathbf{x}) &= \underset{y}{\operatorname{argmax}}\, P(y, \mathbf{x}) / P(\mathbf{x}) \\ &= \underset{y}{\operatorname{argmax}}\, P(y, \mathbf{x}). \end{aligned}$$

- assuming that the attributes are independent given the class

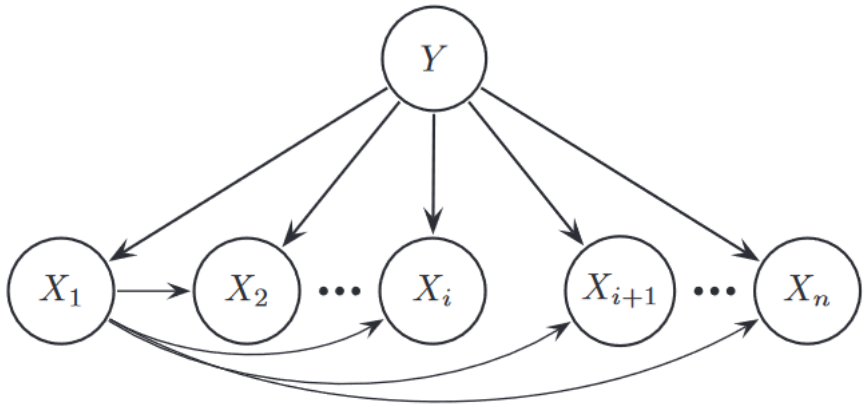
$$\hat{P}(y, \mathbf{x}) = \hat{P}(y) \prod_{i \in N} \hat{P}(x_i \mid y),$$

Semi naïve Bayes (SNB)

- besides the class, SNB allows dependence on some attributes

$$\hat{P}(y, \mathbf{x}) = \hat{P}(y) \prod_{i \in N} \hat{P}(x_i \mid y, \pi(x_i)),$$

- Example: 1-dependence estimator (ODE), where X_1 is “super-parent”

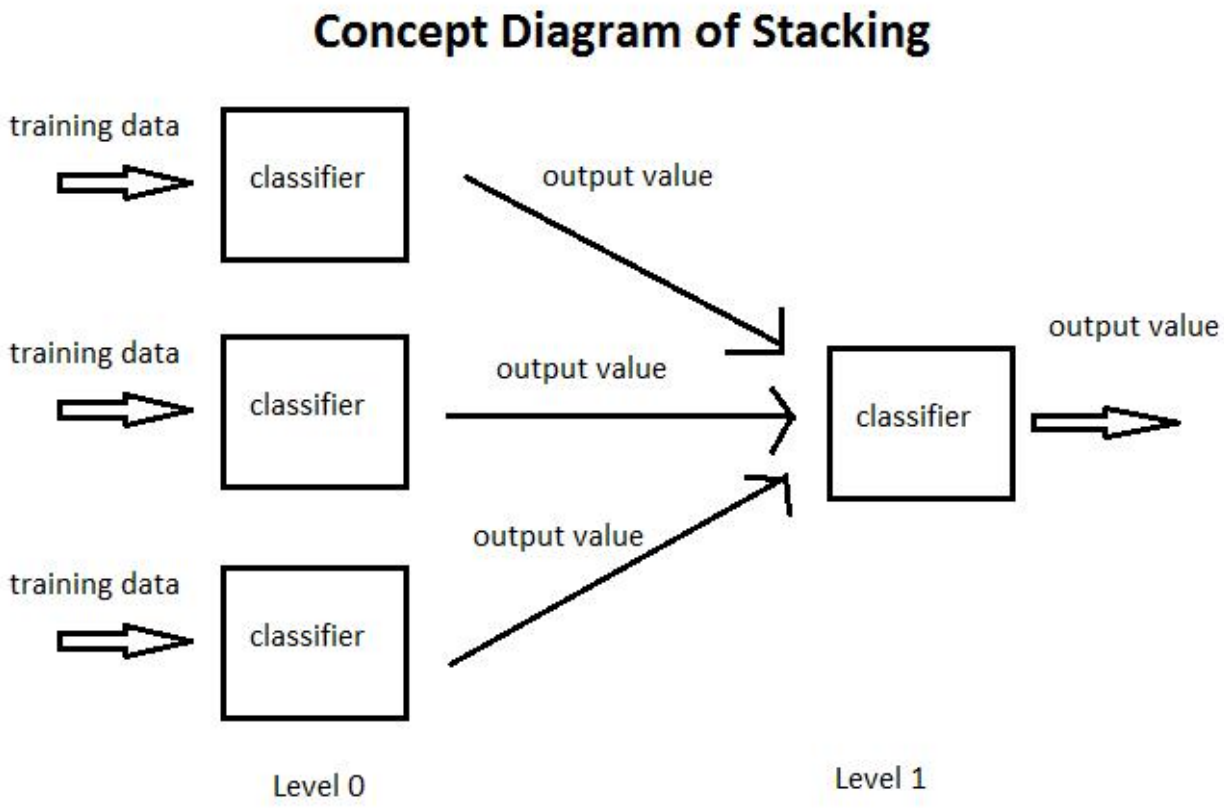


AODE ensemble

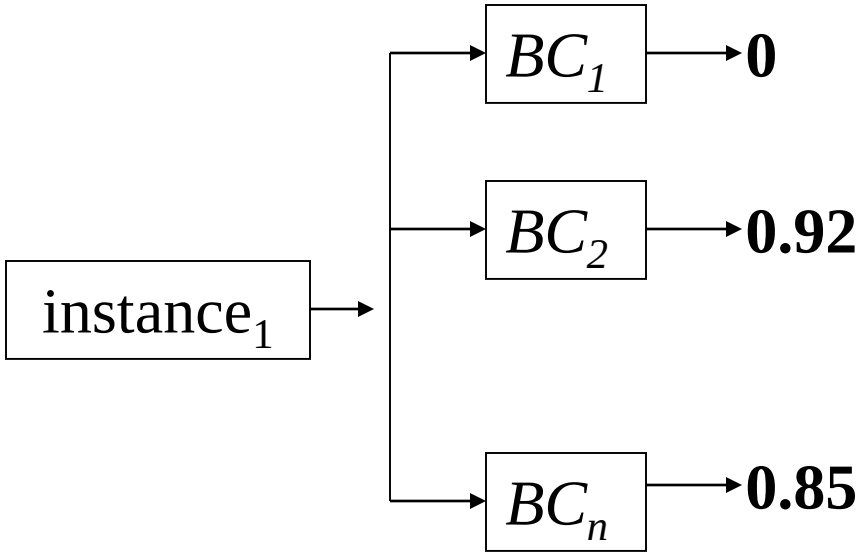
- Averaged One-Dependence Estimator (AODE) (Webb et al. 2005)
- SPODE: Super-Parent One Dependence Estimator – Semi naive Bayes where attributes are dependent on class and one more attribute
- AODE is an ensemble of SPODE classifiers, where all attributes in turn are used in SPODE classifier and their results are averaged
- Compared to naive Bayes, it has higher variance but lower bias

Stacking

- A method to combine heterogeneous predictors
- Predictions of base learners (level-0 models) are used as input for meta learner (level-1 model)
- Base learners are usually different learning schemes

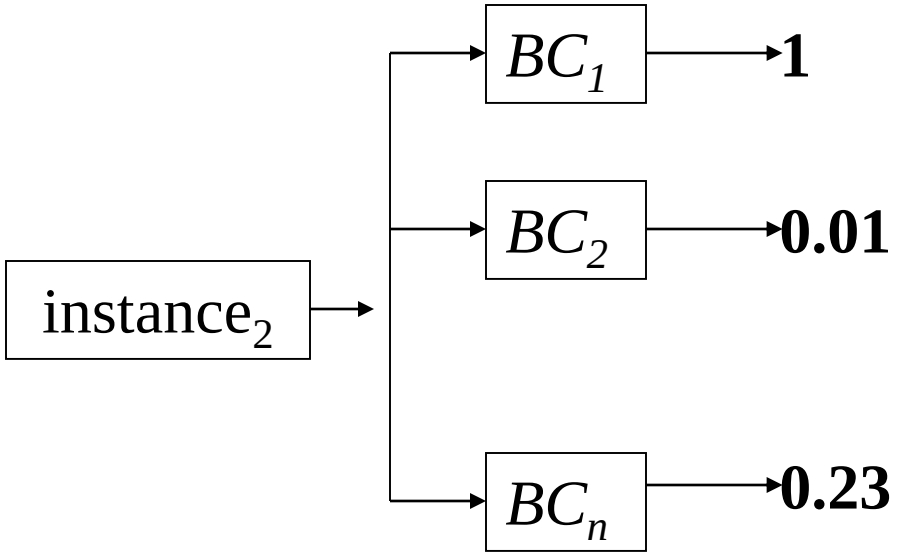


Stacking scheme



meta instances	BC_1	BC_2	…	BC_n	*Class*
instance$_1$	**0**	**0.92**		**0.85**	**1**

Stacking



meta instances	BC_1	BC_2	…	BC_n	*Class*
instance$_1$	**0**	**0.92**		**0.85**	**1**
instance$_2$	**1**	**0.01**		**0.23**	**0**

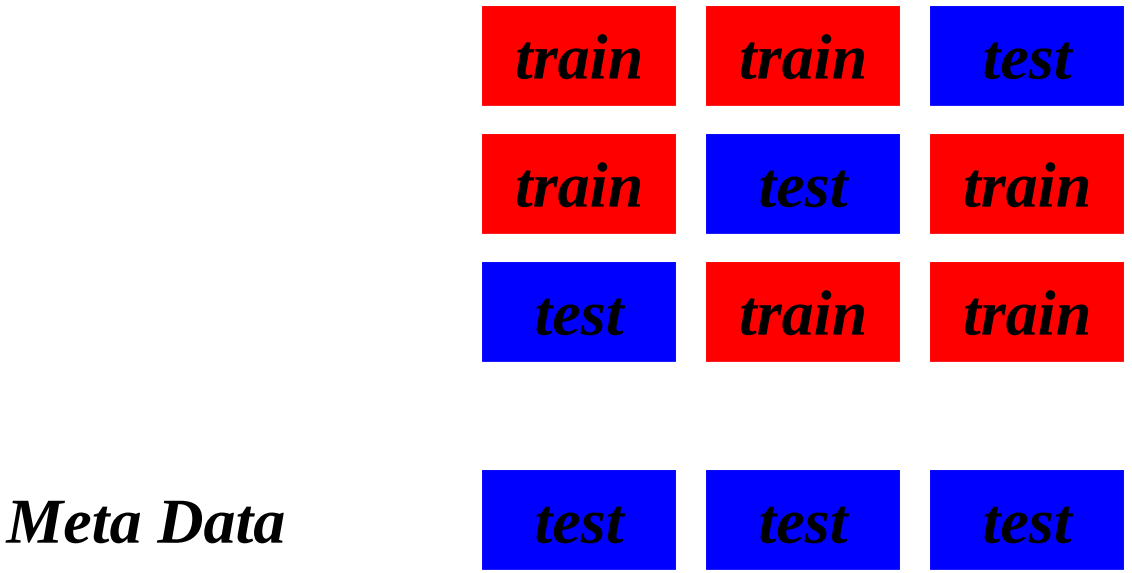
Stacking

Meta Classifier

meta instances	BC_1	BC_2	…	BC_n	*Class*
$instance_1$	**0**	**0.92**		**0.85**	**1**
$instance_2$	**1**	**0.01**		**0.23**	**0**

Actual stacking

- Predictions on the training data can’t be used to generate data for level-1 model! Why not?
- 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!



Stacking meta-learner

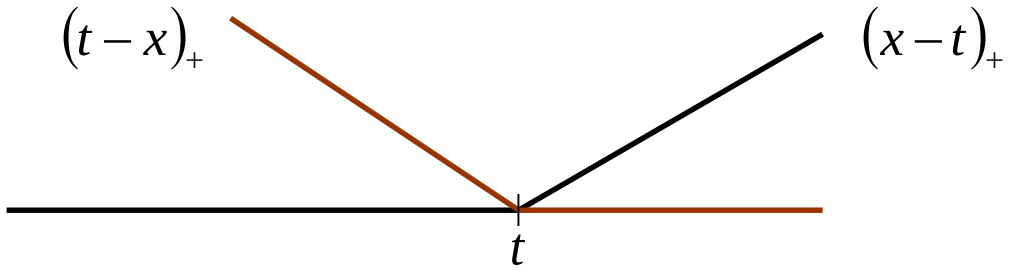
- Which algorithm to use to generate meta learner?
- In principle, any learning scheme can be applied
- For level-1 classifier Ting & Witten (1999) recommend multiple response linear regression (MRLE, note this is a regressor)
  - a classification problem with C classes is transformed into C linear regression problems, where response for problem *i* is 1 if the class equals *i*, otherwise it is 0
  - to classify a new instance employ all *C* linear models, the prediction with highest value is selected as the output

MARS - Multivariate Adaptive Regression Splines

- Generalization of stepwise linear regression
- Modification of trees to improve regression performance
- Able to capture additive structure
- Not tree-based

MARS base models

- Additive model with adaptive set of basis vectors
- Basis built up from simple piecewise linear functions



- Set "*C*" represents candidate set of linear splines, with "knees" at each data point X_j.
- Models are built with elements from C or their products.

$$C = \left\{ (X_j - t)_+, (t - X_j)_+ \right\}_{t \in \{x_{1j}, x_{2j}, \ldots x_{Nj}\} j=1,2,\ldots,p}$$

- Basis collections C: |C| = 2 * N * p

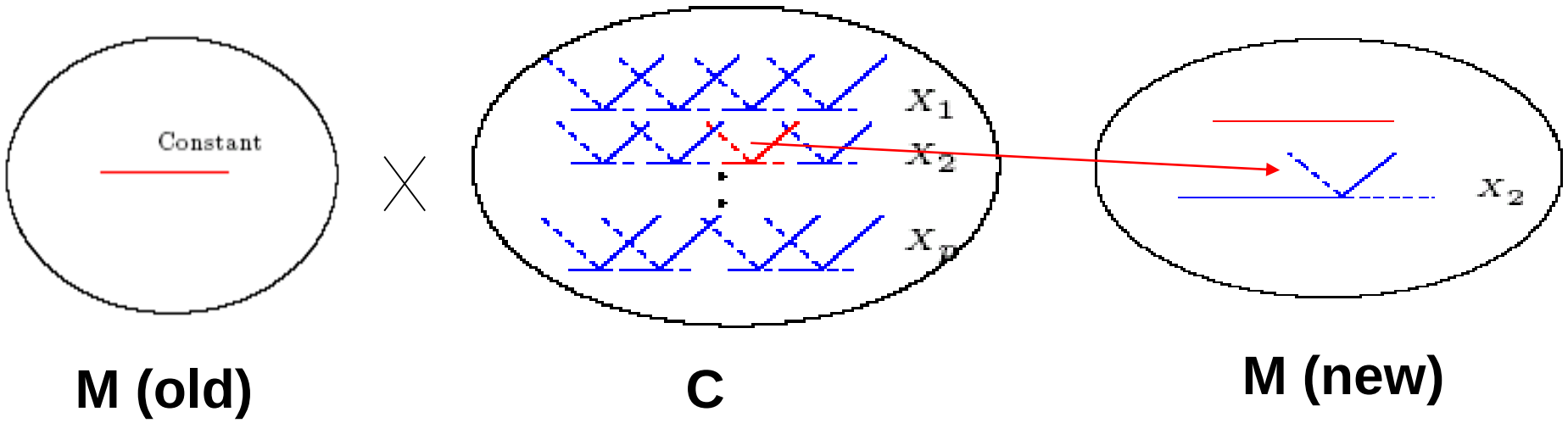
MARS procedure

Model has the form: $f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X)$

1. Given a choice for the h_m, the coefficients β are chosen by the standard linear regression.
2. Start with $h_0(X) = 1$
   All functions in *C* are candidate functions.
3. At each stage, consider as a new basis function pair all products of a function h_m in the model set *M*, with one of the reflected pairs in *C*.
   $\beta_{M+1} h_l(X) \cdot (X_j - t)_+ + \beta_{M+2} h_l(X) \cdot (t - X_j)_+, h_l \in M$
4. We add to the model terms of the form:

   $h_m(X) \cdot (t - X_j)_+$ $\qquad$ $h_m(X) \cdot (X_j - t)_+$

MARS, step 1



- On each step, add the term, which reduces residual error most, into *M*
- Repeat steps (until, e.g., *|M|* >= threshold)

MARS, choosing number of terms

- Large models can overfit.
- Backward deletion procedure: delete terms which cause the smallest increase in residual squared error, to get a sequence of models.
- Pick Model using Generalized Cross Validation:

$$GCV(\lambda) = \frac{\sum_{i=1}^{N} \left(y_i - \hat{f}(x_i)\right)^2}{(1 - M(\lambda)/N)^2}$$

- $M(\lambda)$ is the effective number of parameters in the model.
  C=3, r is the number of basis vectors, and *K* knots

$$M(\lambda) = r + cK$$

- Choose the model which minimizes $GCV(\lambda)$

MARS summary

- Basis functions operate locally
- Forward modeling is hierarchical, multiway products are built up only from existing terms
- Each input appears only once in each product
- Useful option is to set limit on order of operations. Limit of two allows only pairwise products. Limit of one results in an additive model