

# Ensemble methods and Feature selection

Pierre Geurts & Louis Wehenkel

Institut Montefiore, University of Liège, Belgium



ELEN062-1

Introduction to Machine Learning

November 15, 2023

# Part I

## Ensemble methods

- ① Averaging techniques
- ② Boosting techniques
- ③ Other ensemble approaches
- ④ Conclusions on ensemble methods

## Reminder: bias/variance decomposition

$$E_{LS} \left\{ E_{y|\underline{x}} \left\{ (y - \hat{y}(\underline{x}))^2 \right\} \right\} = \text{noise}(\underline{x}) + \text{bias}^2(\underline{x}) + \text{variance}(\underline{x})$$

- $\text{noise}(\underline{x}) = E_{y|\underline{x}} \left\{ (y - h_B(\underline{x}))^2 \right\}$ : quantifies how much  $y$  varies from  $h_B(\underline{x}) = E_{y|\underline{x}}\{y\}$  (the Bayes model).
- $\text{bias}^2(\underline{x}) = (h_B(\underline{x}) - E_{LS}\{\hat{y}(\underline{x})\})^2$ : measures the error between the Bayes model and the average model.
- $\text{variance}(\underline{x}) = E_{LS} \left\{ (\hat{y} - E_{LS}\{\hat{y}(\underline{x})\})^2 \right\}$ : quantifies how much  $\hat{y}(\underline{x})$  varies from one learning sample to another.

# Reminder: bias and variance reduction techniques

- ▶ In the context of a given method:
  - Adapt the learning algorithm to find the best trade-off between bias and variance.
  - Not a panacea but the least we can do.  
**Example:** pruning, weight decay.
- ▶ Ensemble methods:
  - Change the bias/variance trade-off.
  - Universal but destroys some features of the initial method.  
**Example:** bagging, boosting.

# Ensemble methods

They combine the predictions of several models built with a learning algorithm in order to improve with respect to the use of a single model.

There exist two main families:

- ▶ **Averaging** techniques: they grow several models independently and simply average their predictions. They mainly decrease **variance**.  
**Example:** bagging, random forests.
- ▶ **Boosting** techniques: they grow several models sequentially. They mainly decrease **bias**.  
**Example:** adaboost, gradient boosting.

- ① Averaging techniques
  - Bagging
  - Random Forests
  - Ambiguity decomposition
- ② Boosting techniques
- ③ Other ensemble approaches
- ④ Conclusions on ensemble methods

## Bagging (i)

Suppose that we can generate several learning samples  $LS_i$  from the original data distribution  $P(\underline{x}, y)$ .

Let us study the following algorithm:

- Draw  $T$  learning samples  $\{LS_1, LS_2, \dots, LS_T\}$
- Learn a model  $\hat{y}_{LS_i}$  from each  $LS_i$
- Compute the average model  $\hat{y}_{ens}(\underline{x}) = \frac{1}{T} \sum_{i=1}^T \hat{y}_{LS_i}(\underline{x})$

How do the bias and variance of this algorithm relate to that of the original algorithm?



## Bagging (ii)

The bias/variance decomposition is given by:

$$E_{LS}\{\text{Err}(\underline{x})\} = E_{y|\underline{x}} \left\{ (y - h_B(\underline{x}))^2 \right\} + (h_B(\underline{x}) - E_{LS}\{\hat{y}(\underline{x})\})^2 \\ + E_{LS} \left\{ (\hat{y}(\underline{x}) - E_{LS}\{\hat{y}(\underline{x})\})^2 \right\}$$

Its **bias** is the **same** as the original algorithm:

$$E_{LS_1, \dots, LS_T} \{\hat{y}_{ens}(\underline{x})\} = \frac{1}{T} \sum_i E_{LS_i} \{\hat{y}_{LS_i}(\underline{x})\} \\ = E_{LS} \{\hat{y}_{LS}(\underline{x})\}$$

Its **variance** is **divided** by  $T$ :

$$E_{LS_1, \dots, LS_T} \left\{ (\hat{y}_{ens}(\underline{x}) - E_{LS_1, \dots, LS_T} \{\hat{y}_{ens}(\underline{x})\})^2 \right\} \\ = \frac{1}{T} E_{LS} \left\{ (\hat{y}(\underline{x}) - E_{LS} \{\hat{y}(\underline{x})\})^2 \right\}$$

⇒ The mean square error **decreases**.

## Bagging (iii)

In practice, one cannot draw several  $LS_i$ :  $P(\underline{x}, y)$  is **unknown**.

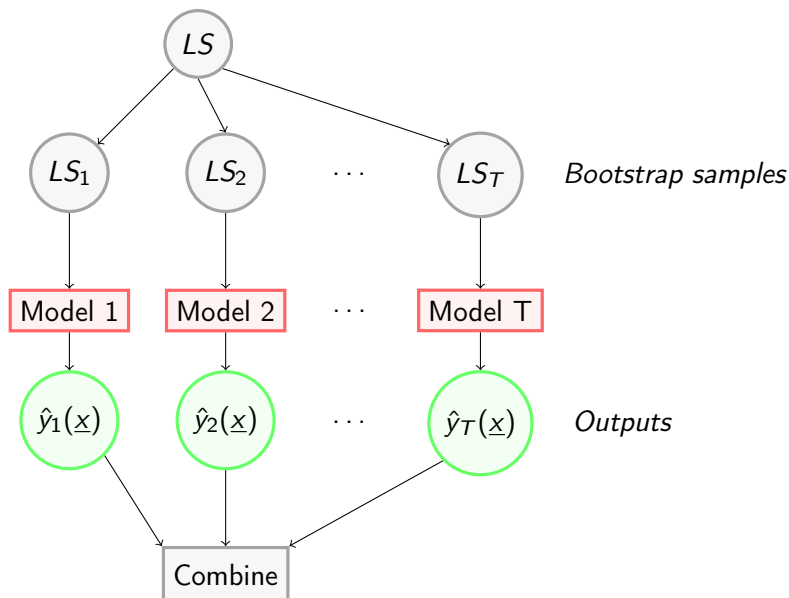
⇒ **Idea**: use bootstrap sampling to generate several learning samples.

This is the idea behind **bagging** (bootstrap **aggregating**):

- Draw  $T$  bootstrap samples  $\{B_1, B_2, \dots, B_T\}$  from  $LS$
- Learn a model  $\hat{y}_{B_i}$  from each  $B_i$
- Build the average model  $\hat{y}_{ens}(\underline{x}) = \frac{1}{T} \sum_{i=1}^T \hat{y}_{B_i}(\underline{x})$

Variance is **reduced** but bias **increases** a bit (because the effective size of a bootstrap sample is about 30% smaller than the original  $LS$ ).

# Bagging (iv)



## Bagging (v)

For a regression problem:

$$\hat{y}_{ens}(\underline{x}) = \frac{1}{T}(\hat{y}_1(\underline{x}) + \hat{y}_2(\underline{x}) + \dots + \hat{y}_T(\underline{x}))$$

For a classification problem, instead, take the majority class in

$$\{\hat{y}_1(\underline{x}), \hat{y}_2(\underline{x}), \dots, \hat{y}_T(\underline{x})\}$$

## Bagging (vi)

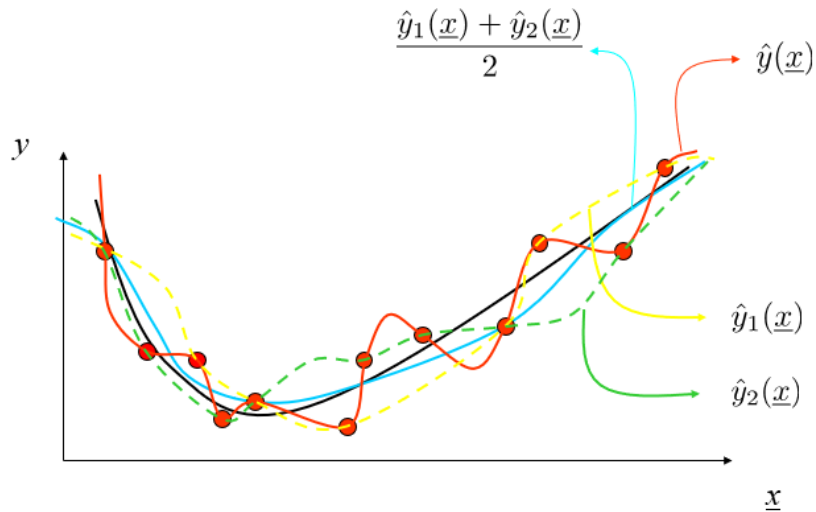
Usually, bagging reduces very much the variance without increasing too much the bias.

Method	E	Bias	Variance
3 Test regression Tree	14.8	11.1	3.7
Bagged (T=25)	11.7	10.7	1.0
Full regression Tree	10.2	3.5	6.7
Bagged (T=25)	5.3	3.8	1.5

Application of bagging to regression trees (on Friedman's problem).

⇒ Strong variance reduction without increasing the bias (although the model is much more complex than a single tree).

# Bagging (vii)



## Other averaging techniques

Another approach is the *perturb and combine* paradigm:

1. Perturb the data/learning algorithm to obtain several models that are good on the learning sample
2. Combine the predictions of these models

Usually, these methods decrease the variance (because of **averaging**) but slightly increase the bias (because of the **perturbation**).

**Examples:**

- Bagging perturbs the learning sample
- Learn several neural networks with random initial weights

Method	E	Bias	Variance
MLP (10-10)	4.6	1.4	3.2
Average of 10 MLPs	2.0	1.4	0.6

- Random forests

## Random forests (i)

Random forests is a type of *perturb and combine* algorithm specifically designed for trees.

It combines **bagging** and **random attribute subset selection**:

- Build the tree from a bootstrap sample
- Instead of choosing the best split among all attributes, select the best split among a **random** subset of  $k$  attributes.

*Note:* It is equivalent to bagging when  $k$  is equal to the number of attributes.

There is a bias/variance trade-off with  $k$ : the smaller the  $k$ , the greater the reduction of variance but the higher the increase of bias.



## Random forests (ii)

Method	E	Bias	Variance
Full regression Tree	10.2	3.5	6.7
Bagging ( $k = 10$ )	5.3	3.8	1.5
Random forests ( $k = 7$ )	4.8	3.8	1.0
Random forests ( $k = 5$ )	4.9	4.0	0.9
Random forests ( $k = 3$ )	5.6	4.7	0.9

Application to our illustrative problem.

Another advantage is that it decreases **computing times** with respect to bagging since only a subset of all attributes is considered when splitting a node.

## Ambiguity decomposition (i)

Let us assume  $T$  models  $\{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_T\}$  and their average

$$\hat{y}_{ens}(\underline{x}) = \frac{1}{T} \sum_i \hat{y}_i(\underline{x})$$

We have the following decomposition:

$$\begin{aligned} \frac{1}{T} \sum_i E_{y|\underline{x}} \left\{ (y - \hat{y}_i(\underline{x}))^2 \right\} &= E_{y|\underline{x}} \left\{ (y - \hat{y}_{ens}(\underline{x}))^2 \right\} \\ &\quad + \frac{1}{T} \sum_i (y_i(\underline{x}) - \hat{y}_{ens}(\underline{x}))^2 \end{aligned}$$

meaning that

$$\begin{aligned} E_{y|\underline{x}} \left\{ (y - \hat{y}_{ens}(\underline{x}))^2 \right\} &= \frac{1}{T} \sum_i E_{y|\underline{x}} \left\{ (y - \hat{y}_i(\underline{x}))^2 \right\} \\ &\quad - \frac{1}{T} \sum_i (y_i(\underline{x}) - \hat{y}_{ens}(\underline{x}))^2 \end{aligned}$$

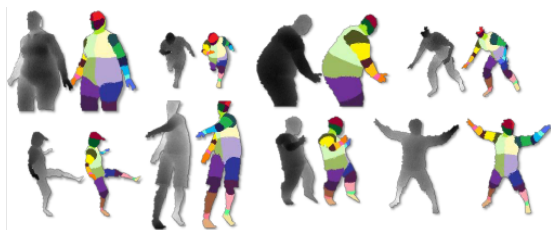
## Ambiguity decomposition (ii)

Hence, the average model is **always** better than the individual models in the mean.

However, this is **not** true in classification!

## Application: Kinect

An ensemble of randomized decision trees is used in Microsoft's Xbox Kinect for body part labelling.



Source: [1]

Each sample corresponds to a single pixel and is described by depth differences between neighbouring pixels.

The final model is implemented on GPU to get very fast predictions (200 frames per second).

# Outline

- ① Averaging techniques
- ② Boosting techniques
  - Adaboost
  - Residual fitting
  - Gradient boosting
  - Bias/variance trade-off
- ③ Other ensemble approaches
- ④ Conclusions on ensemble methods

# Motivation

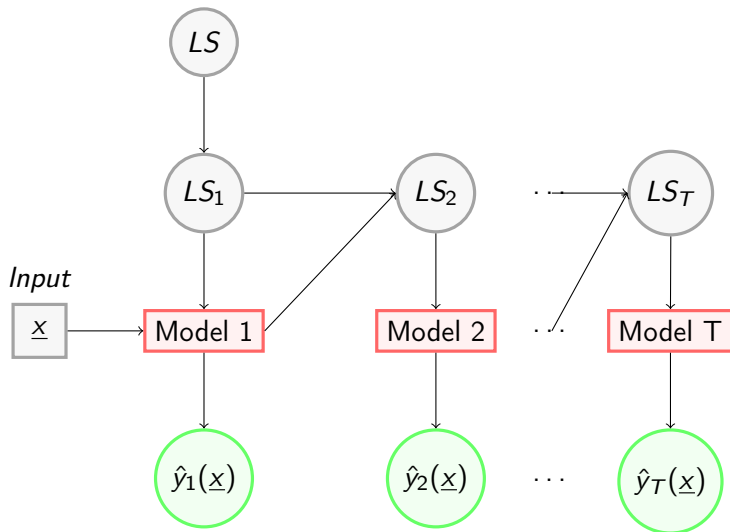
The motivation of boosting is to combine the outputs of many “weak” models to produce a powerful ensemble of models.

*Note:* A “weak” model is a model that has a high bias (strictly, in classification, a model slightly better than random guessing).

Differences with previous ensemble methods:

- Models are built **sequentially** on modified versions of the data.
- The predictions of the models are combined through a **weighted** sum/vote.

# Boosting methods (i)



## Boosting methods (ii)

Regression:

$$\hat{y}(\underline{x}) = \beta_1 \hat{y}_1(\underline{x}) + \beta_2 \hat{y}_2(\underline{x}) + \dots + \beta_T \hat{y}_T(\underline{x})$$

Classification:

$\hat{y}(\underline{x}) =$  majority class in  $\{\hat{y}_1(\underline{x}), \dots, \hat{y}_T(\underline{x})\}$  according to  $\{\beta_1, \dots, \beta_T\}$ .



## Boosting methods (iii)

Many boosting methods exist.

First invented by computer scientists (Freund and Schapire, 1995) and then generalized by statisticians (e.g., Friedman, 2001).

In this lecture:

- ▶ Adaboost: specific method for classification
- ▶ Residual fitting: specific method for regression
- ▶ Gradient boosting: a generic method that can accommodate any loss function.

## Adaboost: main principle

Assume that the learning algorithm accepts weighted objects

$$\{(\underline{x}_1, y_1, w_1), (\underline{x}_2, y_2, w_2), \dots, (\underline{x}_N, y_N, w_N)\}$$

*Note:* This is the case of many learning algorithms:

- With trees, take the weights into account when counting objects
- In neural networks, minimize the weighted squared error

At each step, Adaboost **increases** the weights of cases from the learning sample being **misclassified** by the last model. Therefore, the algorithm focuses on the difficult cases from the learning sample.

In the weighted majority vote, Adaboost gives higher influence to the more accurate models.

# Adaboost: algorithm

- ▶ Input: a learning algorithm and a learning sample

$$\{(\underline{x}_i, y_i) : i = 1, \dots, N\}$$

- ▶ Initialize the weights

$$w_i = \frac{1}{N}, i = 1, \dots, N$$

- ▶ For  $t = 1$  to  $T$ :

1. Build a model  $\hat{y}_t(\underline{x})$  with the learning algorithm using weights  $w_i$
2. Compute the weighted error

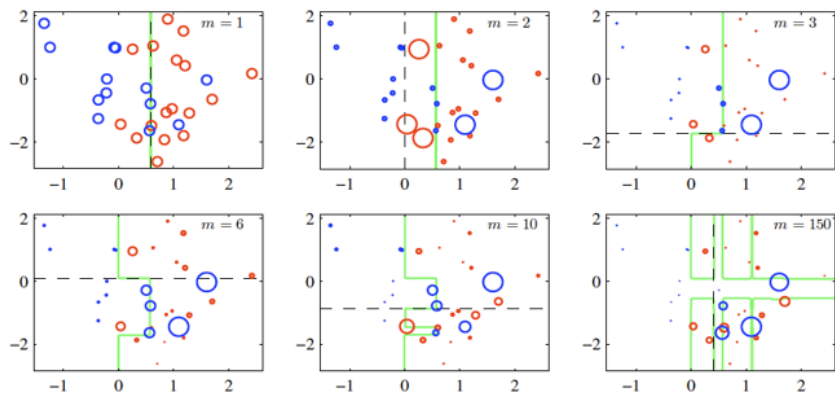
$$err_t = \frac{\sum_i w_i I(y_i \neq \hat{y}_t(\underline{x}_i))}{\sum_i w_i}$$

3. Compute  $\beta_t = \log\left(\frac{1-err_t}{err_t}\right)$
4. Change the weights according to

$$w_i \leftarrow w_i \exp[\beta_t I(y_i \neq \hat{y}_t(\underline{x}_i))]$$

5. Normalize the weights such that  $\sum_i w_i = 1$

# Adaboost: illustration



**Figure 14.2** Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes. Each figure shows the number  $m$  of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the  $m = 1$  base learner are given greater weight when training the  $m = 2$  base learner.

Source: [2]

# Residual fitting: algorithm

(a specific boosting algorithm for least-square regression)

- ▶ Input: a learning sample  $\{(\underline{x}_i, y_i) : i = 1, \dots, N\}$
- ▶ Initialize the model:

$$\hat{y}(\underline{x}) = \hat{y}_0(\underline{x}) = \frac{1}{N} \sum_i y_i$$

- ▶ For  $t = 1$  to  $T$ :
  1. For  $i = 1$  to  $N$ , compute the residuals w.r.t the current  $\hat{y}$ :

$$r_i \leftarrow y_i - \hat{y}(\underline{x}_i)$$

2. Build a regression model  $\hat{y}_t$  from the new learning sample

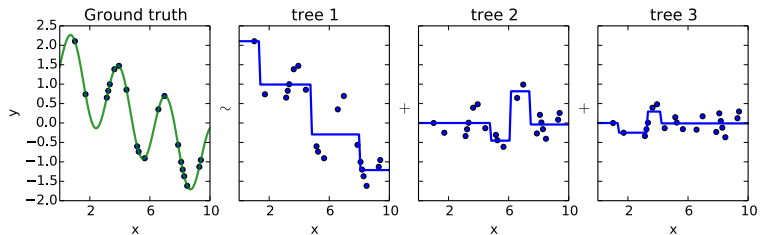
$$\{(\underline{x}_i, r_i) : i = 1, \dots, N\}$$

3. Update the model:  $\hat{y}(\underline{x}) \leftarrow \hat{y}(\underline{x}) + \hat{y}_t(\underline{x})$

- ▶ Return the model  $\hat{y}(\underline{x})$ .

# Residual fitting: illustration

With regression trees:



[Source: Louppe, 2014]

# A generic boosting algorithm

**Goal:** Find  $\hat{y}(\underline{x}) = \beta_1 \hat{y}_1(\underline{x}) + \beta_2 \hat{y}_2(\underline{x}) + \dots + \beta_T \hat{y}_T(\underline{x})$  that minimizes  $\sum_{i=1}^N L(y_i, \hat{y}(\underline{x}_i))$ .

Forward **stage-wise additive** modeling:

1. Initialize  $\hat{y}(\underline{x}) = 0$
2. For  $t = 1$  to  $T$ :
  - 2.1 Compute  $(\beta_t, \hat{y}_t) = \arg \min_{\beta, \hat{y}'} \sum_i L(y_i, \hat{y}(\underline{x}_i) + \beta \hat{y}'(\underline{x}_i))$
  - 2.2 Set  $\hat{y}(\underline{x}) \leftarrow \hat{y}(\underline{x}) + \beta_t \hat{y}_t(\underline{x})$

**Examples:**

- $L(y, y') = (y - y')^2 \Rightarrow$  Least-squares boosting
- $L(y, y') = \exp(-yy') \Rightarrow$  Adaboost (*try to prove it*)

**But** step 2.1 is not easy to solve in general  $\Rightarrow$  replace by a **gradient** step.

General algorithm:

1.  $\hat{y}(\underline{x}) \leftarrow \arg \min_{\gamma} \sum_i L(y_i, \gamma)$
2. For  $t = 1$  to  $T$ :
  - 2.1 For all  $(\underline{x}_i, y_i)$ , compute:

$$r(\underline{x}_i, y_i) = - \left[ \frac{\partial L(y_i, f(\underline{x}_i))}{\partial f(\underline{x}_i)} \right]_{f=\hat{y}}$$

- 2.2  $\hat{y}_t \leftarrow \arg \min_{\hat{y}'} \sum_i (\hat{y}'(\underline{x}_i) - r(\underline{x}_i, y_i))^2$
- 2.3  $\hat{y}(\underline{x}) \leftarrow \hat{y}(\underline{x}) + \mu \hat{y}_t(\underline{x})$ , with  $\mu \in [0, 1]$  a learning rate.

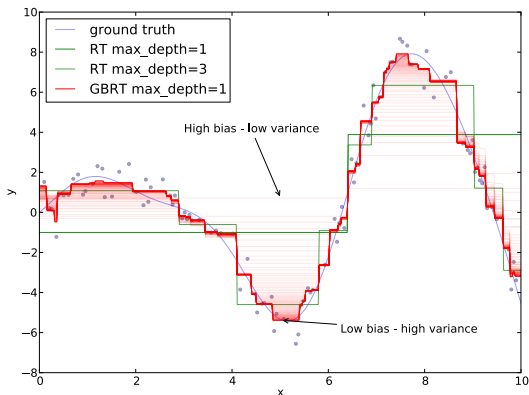
Step 2.2 can be solved with **any** least-square regressor.

Regression trees are the most commonly used base learners (because of low computational cost and good predictive performances).



# Illustration (with regression trees)

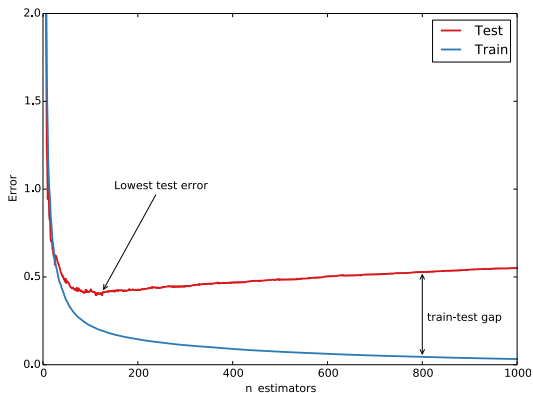
$$Y = \sin(X) + \sin(2 * X) + \epsilon, \text{ avec } X \sim U([0, 10]) \text{ et } \epsilon \sim N(0, 2)$$



Source: <https://github.com/pprett/pydata-gbrt-tutorial>

# Illustration: impact of the number of trees

Unlike Bagging or Random forests, Gradient boosting can overfit with the number of trees



Source: <https://github.com/pprett/pydata-gbrt-tutorial>

# Regularization

Many techniques exist to avoid overfitting:

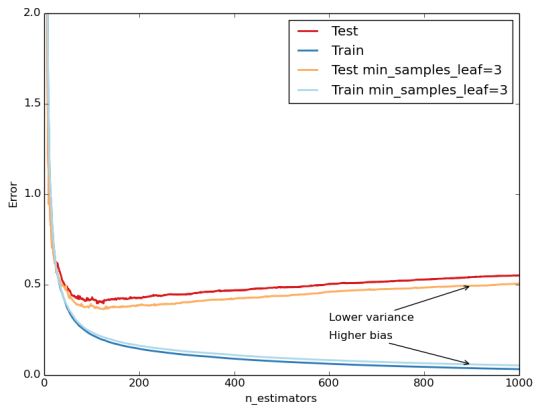
- ▶ **Shrinkage:** use a learning rate  $\mu < 1$ .
- ▶ **Reduce tree complexity:** constrain depth, minimum number of samples per leaf, or number of (test) nodes.
- ▶ **Stochastic gradient boosting:** introduce randomization à la Bagging/Random forests (e.g., feature and sample subsampling)

Each technique introduces a hyper-parameter that regulates some **bias/variance trade-off**.

They **interact** with each other and tuning them requires special care.

# Illustration

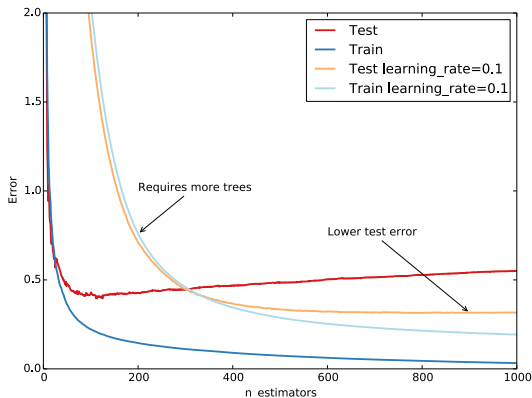
## Impact of constraining leaf sizes



Source: <https://github.com/pprett/pydata-gbrt-tutorial>

# Illustration

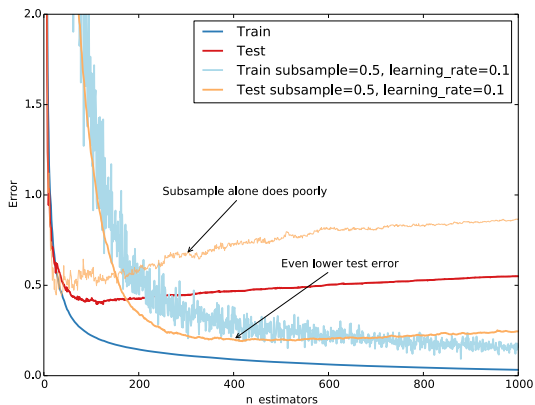
## Impact of learning rate:



Lower learning rates requires more trees.

# Illustration

## Impact of subsampling



Runtime is furthermore reduced by half.

Source: <https://github.com/pprett/pydata-gbrt-tutorial>

# Random forests (RF) vs gradient boosting with trees (GBT)

GBT uses ensembling for reducing **bias** (mostly), while RF uses ensembling for reducing **variance** (only).

GBT's benefits:

- ▶ Often yields better predictive performance than RF.
- ▶ Can be adapted more easily to any (differentiable) loss.
- ▶ Builds smaller trees/ensembles.

RF's benefits:

- ▶ Requires much less hyperparameter tuning. Works out of the box.
- ▶ Embarrassingly parallel.
- ▶ Requires less trick to be implemented efficiently.
- ▶ Better understood theoretically.

## Bias/variance trade-off

Method	E	Bias	Variance
Full regression tree	10.2	3.5	6.7
Regression tree with 1 test	18.9	17.8	1.1
+ GBT ( $T = 50$ )	5.0	3.1	1.9
+ Bagging ( $T = 50$ )	17.9	17.3	0.6
Regression tree with 5 tests	11.7	8.8	2.9
+ GBT ( $T = 50$ )	6.4	1.7	4.7
+ Bagging ( $T = 50$ )	9.1	8.7	0.4

Application to our illustrative problem.

Boosting **reduces** the **bias** but **increases** the **variance**. However, with respect to full trees, it decreases both bias and variance.



A popular **implementation** of Gradient boosting with trees.

Some specificities:

- ▶ Instead of fitting the gradient, it fits a **second order** Taylor expansion of the loss.
- ▶ Instead of using standard regression trees, the impurity measure is **adapted to the loss**.
- ▶ The loss (and thus the impurity) includes a term that penalises for complexity, which results in **automatic tree pruning**:

$$\Omega(\mathcal{T}) = \gamma N_{leaf} + \frac{\lambda}{2} \sum_{l=1}^{N_{leaf}} w_l^2$$

- ▶ Numerical inputs are potentially **binned** and only splits between bins are scored.
- ▶ Support GPU acceleration.

# Outline

- ① Averaging techniques
- ② Boosting techniques
- ③ Other ensemble approaches**
- ④ Conclusions on ensemble methods

## Other ensemble approaches (i)

Bayesian model averaging:

$$P(y | \underline{x}, LS) = \sum_{h \in \mathcal{H}} P(y | h, LS) P(h | LS)$$

with

$$\begin{aligned} P(h | LS) &\propto P(h) P(LS | h) \\ &\propto P(h) \sum_{\theta} P(LS | \theta, h) P(\theta | h) \end{aligned}$$

where:

- $P(h)$  is the prior knowledge about models (e.g. simple models are more probable)
- $P(LS | h)$  is the quality of the fit

## Other ensemble approaches (ii)

### Stacking:

Learn a model to combine the models

- Let  $LS = \{(\underline{x}_i, y_i) \mid i = 1, \dots, N\}$
- Let  $A^t, t = 0, \dots, T$  be  $T + 1$  learning algorithms
- For  $t = 1, \dots, T$ :
  1. Construct a model:  $\hat{y}^t = A^t(LS)$
  2. Compute predictions:  $\hat{y}_i^t = \hat{y}^t(\underline{x}_i)$
- Set  $LS^0 = \{(\underline{x}_i^0, y_i)\}$  with  $\underline{x}_i^0 = (\hat{y}_i^t)_{t=1}^T$
- Return  $\hat{y} = A^0(LS^0)$

# Outline

- ① Averaging techniques
- ② Boosting techniques
- ③ Other ensemble approaches
- ④ Conclusions on ensemble methods
  - Interpretability and efficiency
  - Conclusion

Since we average several models, we lose the interpretability of the combined models and some efficiency.

However:

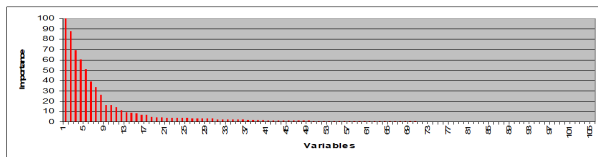
- With trees, we can still use the ensembles to compute variable importance by averaging over all trees. Actually, this even stabilizes the estimates.
- Averaging techniques can be parallelized and boosting type algorithms use smaller trees. Hence, the increase of computing times is not so detrimental.

# Experiments on Golub's microarray data

- ▶ Input: 72 objects, 7129 numerical attributes (gene expressions), 2 classes (ALL and AL)
- ▶ Leave-one-out error with several variants

Method	Error
1 decision tree	22.2%(16/72)
Random forests ( $k = 85$ , $T = 500$ )	9.7%(7/72)
Extra-trees ( $s_{th} = 0.5$ , $T = 500$ )	5.5%(4/72)
Adaboost (1 test node, $T = 500$ )	1.4%(1/72)

- ▶ Variable importance with boosting



# Conclusion

Ensemble methods are very effective techniques to **reduce** bias **and/or** variance. They can transform a not-so-good method to a competitive method in terms of accuracy.

Random forests and gradient boosting with trees are two very competitive methods, with the first easier to use and the second often more accurate.

Interpretability of the model and efficiency of the method are **difficult** to preserve if we want to reduce variance significantly.

There are other ways to tackle the variance/overfitting problem:

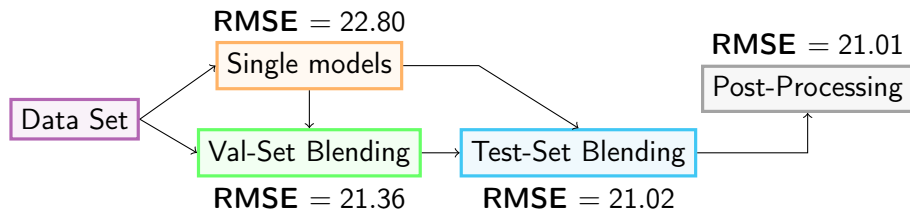
- Bayesian approaches ( $\sim$  averaging techniques)
- Support vector machines, which maintain a low variance by maximizing the classification margin



# Machine learning challenges

Machine learning challenges are commonly won by ensemble solutions:

- ▶ Netflix prize (\$1M reward): best solution combines 107 models obtained from different methods (*stacking*)
- ▶ Yahoo! 2011 KDD Cup (\$5 000 reward): best solution uses two levels of stacking



# Further reading and software

## Further reading:

- ▶ Hastie et al.: 8.7, 8.8, 10.1-4, 15 (not in detail)

## Software:

- ▶ Many ensemble methods are available in common ML libraries (scikit-learn, R)
- ▶ Boosting:
  - XGBoost: <http://xgboost.readthedocs.io/>
  - Light-GBM: <https://github.com/Microsoft/LightGBM>
- ▶ Demos:
  - [http://arogozhnikov.github.io/2016/07/05/gradient\\_boosting\\_playground.html](http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html)
  - <https://cs.stanford.edu/~karpathy/svmjs/demo/demoforest.html>

## Part II

### Feature selection

# Outline

- ① Motivation and formalization
- ② Filter techniques
- ③ Embedded techniques
- ④ Wrapper methods
- ⑤ Selection bias
- ⑥ Further reading

Feature selection is a technique to **reduce** the number of features used by the learning algorithm.

It has several advantages:

- Avoid overfitting and improve model performance
- Improve interpretability
- Provide faster and more cost-effective models
- Reduce overall computing times, if the feature selection technique is fast

# Feature selection vs Ranking

## Feature selection:

Find a small (or the smallest) subset of features that maximizes accuracy.

## Feature ranking:

Sort the variables according to their relevance at predicting the output.

There exist techniques in both families.

*Note:* Feature selection can be obtained from feature ranking, e.g. by selecting the top  $k$  features in a ranking.

# Formalization

Let  $Y$  denote the class variable and  $V = \{X_1, \dots, X_p\}$  the set of input variables.

A feature  $X_i$  is:

- **strongly relevant** iff  $P(Y | X_i, V \setminus X_i) \neq P(Y | V \setminus X_i)$
- **weakly relevant** iff it is not strongly relevant and  $P(Y | X_i, S) \neq P(Y | S)$  for some subset  $S \subset V$
- **irrelevant** otherwise

A subset  $M \subseteq V$  of variables is a **Markov boundary** for  $Y$  if it is minimal and  $P(Y | M, V \setminus M) = P(Y | M)$ : features in  $M$  are either weakly or strongly relevant. They are all strongly relevant when the distribution  $P$  is strictly positive.

Feature selection is often formulated as finding a Markov boundary  $M$  for  $Y$ .

*Note:* All variables in a Markov boundary do not necessarily appear in the Bayes model (depending on the loss function).

There are three main approaches:

- ▶ **Filter techniques:** a priori selection of the variables, *i.e.* independently of the supervised learning algorithm.
- ▶ **Embedded techniques:** feature selection is embedded in the learning algorithm.
- ▶ **Wrapper methods:** use cross-validation to find the optimal set of features for a given algorithm.



# Filter techniques

**Idea:** associate a relevance score to each feature and remove the low-scoring ones.

Univariate scoring (often used):

- ▶ Any score measures used in decision trees
- ▶ Statistical test (t-test, chi-square, etc.)

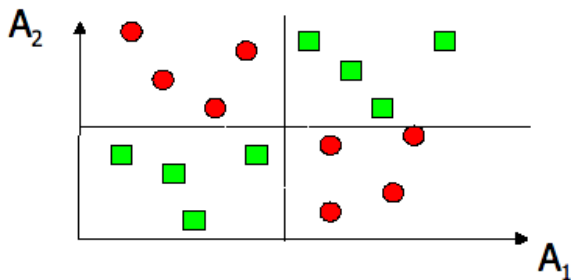
Multivariate scoring:

- ▶ Relief
- ▶ Markov blanket filter
- ▶ Decision trees
- ▶ ...

The optimal number of features can be determined by cross-validation.

## Univariate scoring vs Multivariate scoring

Each feature is useless alone (low univariate scoring) but together they perfectly explain the classification.



# Conclusions

## Advantages:

- ▶ Univariate scoring is fast and scalable.
- ▶ Independent of the supervised learning algorithm.

## Drawbacks:

- ▶ It ignores the supervised learning algorithm.
- ▶ Univariate scoring ignores feature dependencies.
- ▶ Multivariate scoring is slower than univariate scoring.

# Embedded techniques

Some supervised learning methods embed feature selection. The search for an optimal subset of features is built into the learning algorithm.

## Examples:

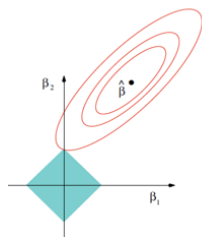
- Decision tree node splitting is a feature selection technique.
- Tree ensemble variable importance measures.
- Absolute weights in a linear SVM model:

$$\hat{y}(\underline{x}) = \text{sgn}\left(\sum_i w_i x_i + b\right)$$

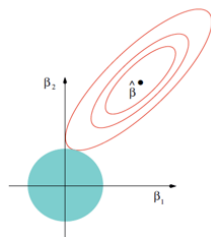
# LASSO

Linear model learned with L1 penalization.

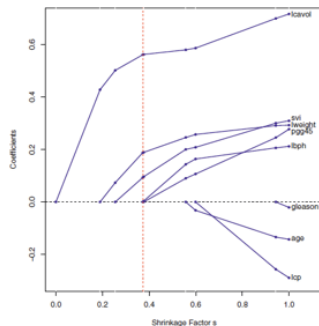
$$\min_{\beta} \sum_{i=1}^N (y_i - (\beta_0 + \sum_j \beta_j x_j))^2 + \lambda \sum_j |\beta_j|$$



LASSO



Ridge



Influence of  $\lambda$ .

# Conclusions

## Advantages:

- ▶ Usually computationally efficient.
- ▶ Well integrated within the learning algorithm.
- ▶ Multivariate.

## Drawbacks:

- ▶ Specific to the learning algorithm.

# Wrapper methods

**Idea:** Try to find a subset of features that maximizes the quality of the model induced by the learning algorithm. The quality is assessed by cross-validation.

As the number of subsets of  $p$  features is  $2^p$ , all subsets can usually **not** be evaluated, and heuristics are necessary.

Many approaches exist:

- Forward/Backward selection: add (resp. remove) the variable that most decreases (resp. less increases) the error.
- Optimization by genetic algorithms.

## Recursive feature elimination (i)

It is a popular wrapper, especially in bioinformatics.

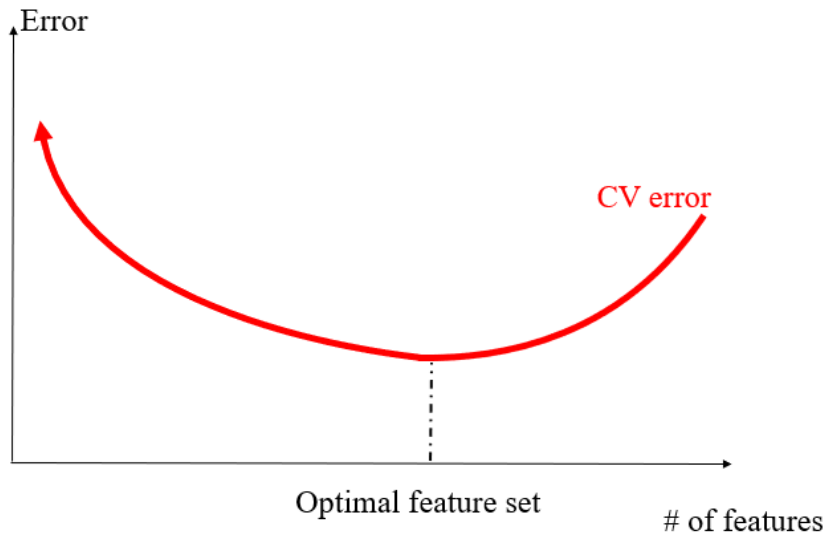
Assume that we have a learning algorithm that can **rank** the features (e.g. linear SVM, decision trees).

Iterate:

1. Learn a model from the current feature set.
  2. Rank the features with the model.
  3. Remove the feature with the smallest ranking.
- Keep the feature set that yields the lowest cross-validation error.



## Recursive feature elimination (ii)



# Conclusions

## Advantages:

- ▶ Custom-tailored to the learning algorithm.
- ▶ Find interactions and remove redundant variables.

## Drawbacks:

- ▶ Prone to **overfitting**: it is often easy to find a small subset of noisy features that discriminates perfectly the classes.
- ▶ Expensive: we have to build a model for each subset of variables.

## Selection bias (i)

We often see this experiment:

- Select the  $N$  top variables using some filter on the full dataset.
- Evaluate an algorithm that uses these  $N$  variables as inputs by cross-validation (e.g. LOO) on the dataset.

What is wrong with this protocol?

## Selection bias (ii)

A1	A2	...	A1000	Y
-0.86	0.17	...	0	C2
-2.3	-1.2	...	-0.42	C1
-0.37	-0.11	...	-0.64	C1
0.41	0.67	...	-0.8	C2
-0.51	-0.59	...	0.98	C2
-0.25	-0.27	...	-0.68	C1
-0.52	0.23	...	0.11	C1
-1.3	-0.2	...	0.14	C1
0.93	-0.78	...	-0.01	C2
-0.25	-0.29	...	0.69	C2
-0.6	0.92	...	-0.64	C1
0.22	-0.8	...	-0.5	C2
-0.62	0.2	...	0.08	C1
-0.3	0.8	...	0.02	C2
-0.91	0.44	...	-0.57	C1
0.76	0.65	...	-0.08	C1

Dataset composed of 250 objects. Each object has 1 000 features and a single class. Each feature is randomly drawn from  $\mathcal{N}(0, 1)$ , and the class is selected at random.

## Selection bias (iii)

Let us consider two trials:

- Tree bagging without feature selection  $\rightarrow$  10-fold cross-validation error  $\approx 52\%$
- Tree bagging with 20 top features (t-test)  $\rightarrow$  10-fold cross-validation error  $\approx 35\%$

One could conclude that there are 20 interesting variables, and that, from them, one can classify better than at random.

**However**, on a new set of 250 samples, the error is 52%.

## Selection bias (iv)

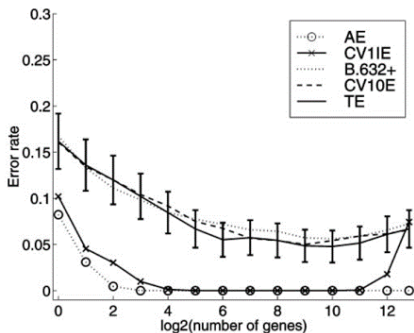
We have both selected the variables **and** tested the model on the basis of the whole training set  $\Rightarrow$  the CV error estimate is too optimistic (overfitting).

A correct protocol to assess the model would be:

- Divide the learning sample into 10 folds.
- For  $i = 1$  to 10:
  1. Remove the  $i$ -th fold from the learning sample.
  2. Select the top 20 variables from the remaining objects.
  3. Learn the model using the 20 variables and the remaining objects.
  4. Test the model on the  $i$ -th fold.

# Selection bias (v)

- ▶ AE = error on the learning set
- ▶ CV1IE = internal LOO
- ▶ CV10E = external 10-fold cross-validation
- ▶ TE = error on an independent test set
- ▶ B.632+ = another unbiased error estimate



Source: [3]

# Outline

- ① Motivation and formalization
- ② Filter techniques
- ③ Embedded techniques
- ④ Wrapper methods
- ⑤ Selection bias
- ⑥ Further reading



## Further reading and software

### Further reading:

- Hastie et al.: chapter 18.
- Guyon and Elisseeff, An introduction to variable and feature selection.

<http://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf>

<http://clopinet.com/fextract-book/IntroFS.pdf>




- Saeys et al. A review of feature selection techniques in bioinformatics.

<http://dx.doi.org/10.1093/bioinformatics/btm344>

### Software:

- ▶ Many feature selection methods (filters, embedded or wrappers) are implemented in common ML libraries (scikit-learn, R).
- ▶ For example: [https://scikit-learn.org/stable/modules/feature\\_selection.html](https://scikit-learn.org/stable/modules/feature_selection.html).

# References

-  Jamie Shotton, Andrew Fitzgibbon, Andrew Blake, Alex Kipman, Mark Finocchio, Bob Moore, and Toby Sharp.  
Real-time human pose recognition in parts from a single depth image.  
In *CVPR*. IEEE, June 2011.  
Best Paper Award.
-  Christopher M Bishop.  
*Pattern recognition and machine learning*.  
springer, 2006.
-  Christophe Ambroise and Geoffrey J McLachlan.  
Selection bias in gene extraction on the basis of microarray gene-expression data.  
*Proceedings of the national academy of sciences*,  
99(10):6562–6566, 2002.