# Lecture 6 – Tree-based methods: Bagging, Boosting and Random Forests

UPPSALA
UNIVERSITET

**Jens Sjölund**
`https://jsjol.github.io/`
Department of Information Technology
Uppsala University

Course webpage

# Summary of Lecture 5 (I/IV)

When **choosing models/methods**, we are interested in how well they will perform when **faced with new unseen data**.

The new data error

$$E_{\text{new}} \triangleq \mathbb{E}_{\star}\left[E(\widehat{y}(\mathbf{x}_{\star};\mathcal{T}), y_{\star})\right]$$

describes how well a method (which is trained using data set $\mathcal{T}$) will perform "in production".

$E$ is for instance mean squared error (regression) or misclassification (classification).

**The overall goal in supervised machine learning is to achieve small $E_{\text{new}}$.**

$E_{\text{train}} \triangleq \frac{1}{n}\sum_{i=1}^{n} E(\widehat{y}(\mathbf{x}_i;\mathcal{T}), y_i)$ is the training data error.
**Not a good estimate of $E_{\text{new}}$.**

# Summary of Lecture 5 (II/IV)

Two methods for estimating $E_{\text{new}}$:

1. **Hold-out validation data approach:** Randomly split the data into a **training set** and a **hold-out validation set**. Learn the model using the training set. Estimate $E_{\text{new}}$ using the hold-out validation set.
2. **$k$-fold cross-validation:** Randomly split the data into $k$ parts (or **folds**) of roughly equal size.
   a) The first fold is kept aside as a validation set and the model is learned using only the remaining $k-1$ folds. $E_{\text{new}}$ is estimated on the validation set.
   b) The procedure is repeated $k$ times, each time a different fold is treated as the validation set.
   c) The average of all $k$ estimates is taken as the final estimate of $E_{\text{new}}$.

# Summary of Lecture 5 (III/IV)

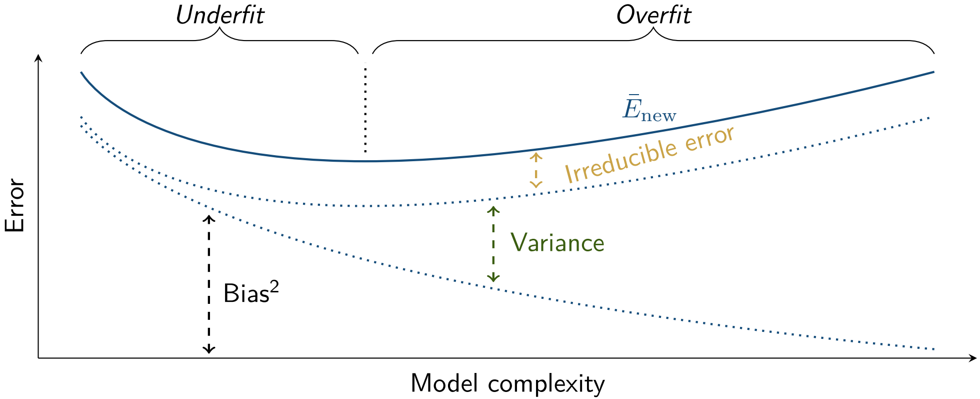
$$\bar{E}_{\text{new}} = \underbrace{\mathbb{E}_\star\left[\left(\bar{f}(\mathbf{x}_\star) - f_0(\mathbf{x}_\star)\right)^2\right]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_\star\left[\mathbb{E}_\mathcal{T}\left[\left(\widehat{y}(\mathbf{x}_\star;\mathcal{T}) - \bar{f}(\mathbf{x}_\star)\right)^2\right]\right]}_{\text{Variance}} + \underbrace{\sigma^2}_{\substack{\text{Irreducible}\\ \text{error}}}$$

where $\quad \bar{f}(\mathbf{x}) = \mathbb{E}_\mathcal{T}\left[\widehat{y}(\mathbf{x}_\star;\mathcal{T})\right]$

- **Bias**: The inability of a method to describe the complicated patterns we would like it to describe.
- **Variance**: How sensitive a method is to the training data.

The more prone a model is to adapt to complicated pattern in the data, the higher the **model complexity** (or model flexibility).

# Summary of Lecture 5 (IV/IV)



Finding a balanced fit (neither over- nor underfit) is called the **the bias-variance tradeoff**.

# Contents – Lecture 6

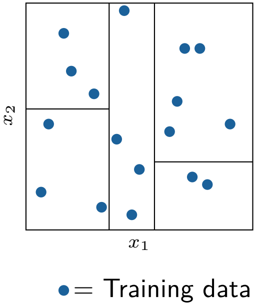
1. Tree-based methods
2. Bagging – *a general variance reduction technique*
3. Random forests
4. Boosting

# Tree-based methods

# The idea behind tree-based methods

In both regression and classification settings we seek a function $\widehat{y}(\mathbf{x})$ which maps the input $\mathbf{x}$ into a prediction.

One **flexible** way of designing this function is to partition the input space into disjoint regions and fit a simple model in each region.



- **Classification:** Majority vote within the region.
- **Regression:** Mean of training data within the region.

# Finding the partition

The key challenge in using this strategy is to find a good partition.

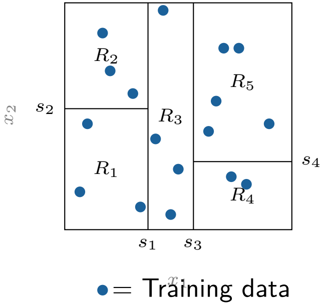
Even if we restrict our attention to seemingly simple regions (e.g. "boxes"), finding an **optimal** partition w.r.t. minimizing the training error is **computationally infeasible!**

Instead, we use a "greedy" approach: **recursive binary splitting**.

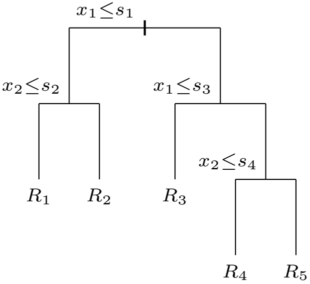
1. Select one input variable $x_j$ and a cut-point $s$. Partition the input space into two half-spaces,
$$\{\mathbf{x} : x_j < s\} \qquad \{\mathbf{x} : x_j \geq s\}.$$
2. Repeat this splitting for each region until some stopping criterion is met (e.g., no region contains more than 5 training data points).

# Recursive binary splitting

Partitioning of input space



Tree representation



# Regression trees

Once the input space is partitioned into $L$ regions, $R_1, R_2, \dots, R_L$ the prediction model is

$$\widehat{y}_\star = \sum_{\ell=1}^{L} \widehat{y}_\ell \mathbb{I}\{\mathbf{x}_\star \in R_\ell\},$$

where $\mathbb{I}\{\mathbf{x}_\star \in R_\ell\}$ is the indicator function

$$\mathbb{I}\{\mathbf{x}_\star \in R_\ell\} = \begin{cases} 1 & \text{if } \mathbf{x}_\star \in R_\ell \\ 0 & \text{if } \mathbf{x}_\star \notin R_\ell \end{cases}$$

and $\widehat{y}_\ell$ is a constant prediction within each region.
For regression trees we use

$$\widehat{y}_\ell = \text{average}\{y_i : \mathbf{x}_i \in R_\ell\}$$

# Recursive binary splitting for a regression tree

Recursive binary splitting is **greedy** - each split is made in order to minimize the loss **without looking ahead** at future splits.

For any $j$ and $s$, define

$$R_1(j,s) = \{\mathbf{x} \,|\, x_j < s\} \qquad \text{and} \qquad R_2(j,s) = \{\mathbf{x} \,|\, x_j \geq s\}.$$

We then seek $(j,s)$ that minimize

$$\sum_{i:\mathbf{x}_i \in R_1(j,s)} \left(y_i - \widehat{y}_1(j,s)\right)^2 + \sum_{i:\mathbf{x}_i \in R_2(j,s)} \left(y_i - \widehat{y}_2(j,s)\right)^2$$

where

$$\widehat{y}_1 = \text{average}\{y_i : \mathbf{x}_i \in R_1(j,s)\}$$
$$\widehat{y}_2 = \text{average}\{y_i : \mathbf{x}_i \in R_2(j,s)\}$$

This optimization problem is easily solved by ”brute force”.

# Classification trees

Classification trees are constructed similarly to regression trees, but with *two differences*.

***Firstly,*** the class prediction for each region is based on the proportion of data points from each class in that region. Let

$$\widehat{\pi}_{\ell m} = \frac{1}{n_\ell} \sum_{i:\mathbf{x}_i \in R_\ell} \mathbb{I}\{y_i = m\}$$

be the proportion of training data points in the $l$th region that belong to the $m$th class. Then we approximate

$$p(y = m \,|\, \mathbf{x}_\star) \approx \sum_{\ell=1}^{L} \widehat{\pi}_{\ell m} \mathbb{I}\{\mathbf{x}_\star \in R_\ell\}$$

# Classification trees

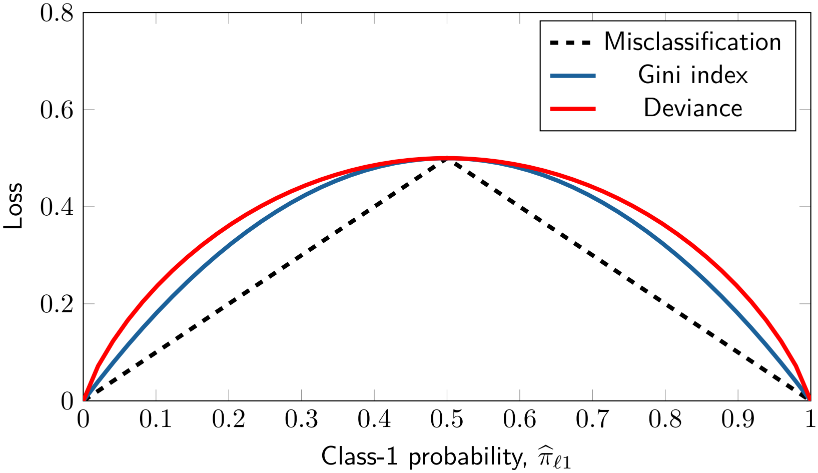
***Secondly,*** the squared loss used to decide the splits needs to be replaced by a measure suitable to categorical outputs.

Three common error measures are,

$$\begin{aligned}\text{Misclassification error:} &\quad 1 - \max_m \widehat{\pi}_{\ell m}\\ \text{Entropy/deviance:} &\quad -\textstyle\sum_{m=1}^{M} \widehat{\pi}_{\ell m} \log \widehat{\pi}_{\ell m}\\ \text{Gini index:} &\quad \textstyle\sum_{m=1}^{M} \widehat{\pi}_{\ell m}(1 - \widehat{\pi}_{\ell m})\end{aligned}$$

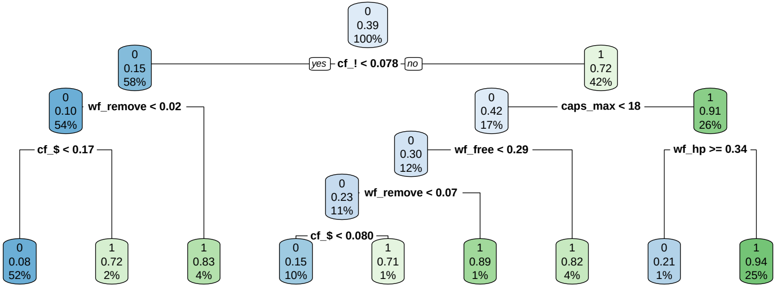
# Classification error measures

For a binary classification problem ($M = 2$)



# *ex)* Spam data

Classification tree for spam data:



| | Tree | LDA |
|---|---|---|
| Test error: | 11.3 % | 10.9 % |

# Improving CART

The performance of (simple) CARTs is often unsatisfactory!

The flexibility/complexity of classification and regression trees (CART) is decided by the tree depth.

- ! To obtain a **small bias** the tree need to be grown deep,
- ! but this results in a **high variance!**

To improve the practical performance:

- ▲ **Pruning –** grow a deep tree (small bias) which is then pruned into a smaller one (reduce variance).
- ▲ **Ensemble methods –** average or combine multiple trees.

# Bagging

# Probability detour - Variance reduction by averaging

Let $z_b$, $b = 1, \dots, B$ be identically distributed random variables with mean $\mathbb{E}[z_b] = \mu$ and variance $\mathrm{Var}[\sigma^2]$. Let $\rho$ be the correlation between distinct variables.

Then,

$$\mathbb{E}\left[\frac{1}{B}\sum_{b=1}^{B} z_b\right] = \mu,$$

$$\mathrm{Var}\left[\frac{1}{B}\sum_{b=1}^{B} z_b\right] = \underbrace{\frac{1-\rho}{B}\sigma^2}_{\text{small for large } B} + \rho\sigma^2.$$

The variance is reduced by averaging (if $\rho < 1$) !

# Bagging (I/II)

For now, assume that we have access to $B$ **independent** datasets $\mathcal{T}^1, \dots, \mathcal{T}^B$. We can then train a separate deep tree $\widehat{y}^b(\mathbf{x})$ for each dataset, $1, \dots, B$.

- Each $\widehat{y}^b(\mathbf{x})$ has a **low bias** but **high variance**
- By averaging

$$\widehat{y}_{\mathsf{bag}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\widehat{y}^b(\mathbf{x})$$

the bias is kept small, but variance is reduced by a factor $B$!

# Bagging (II/II)

**Obvious problem.** We only have access to one training dataset.

**Solution** Bootstrap the data!

- Sample $n$ times with replacement from the original training data $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^n$
- Repeat $B$ times to generate $B$ "bootstrapped" training datasets $\widetilde{\mathcal{T}}^1, \ldots, \widetilde{\mathcal{T}}^B$

For each bootstrapped dataset $\widetilde{\mathcal{T}}^b$ we train a tree $\widetilde{y}^b(\mathbf{x})$. Averaging these,

$$\widetilde{y}^b_{\text{bag}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \widetilde{y}^b(\mathbf{x})$$

is called "bootstrap aggregation", or bagging.

# Bagging - Toy example

*ex)* Assume that we have a training set

$$\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)\}.$$

We generate, say, $B = 3$ datasets by bootstrapping:

$$\begin{aligned}
\widetilde{\mathcal{T}}^1 &= \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_3, y_3)\} \\
\widetilde{\mathcal{T}}^2 &= \{(\mathbf{x}_1, y_1), (\mathbf{x}_4, y_4), (\mathbf{x}_4, y_4), (\mathbf{x}_4, y_4)\} \\
\widetilde{\mathcal{T}}^3 &= \{(\mathbf{x}_1, y_1), (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_2, y_2)\}
\end{aligned}$$

Compute $B = 3$ (deep) regression trees $\tilde{y}^1(\mathbf{x})$, $\tilde{y}^2(\mathbf{x})$ and $\tilde{y}^3(\mathbf{x})$, one for each dataset $\widetilde{\mathcal{T}}^1$, $\widetilde{\mathcal{T}}^2$, and $\widetilde{\mathcal{T}}^3$, and average

$$\tilde{y}_{\text{bag}}(\mathbf{x}) = \frac{1}{3}\sum_{b=1}^{3} \tilde{y}^b(\mathbf{x})$$

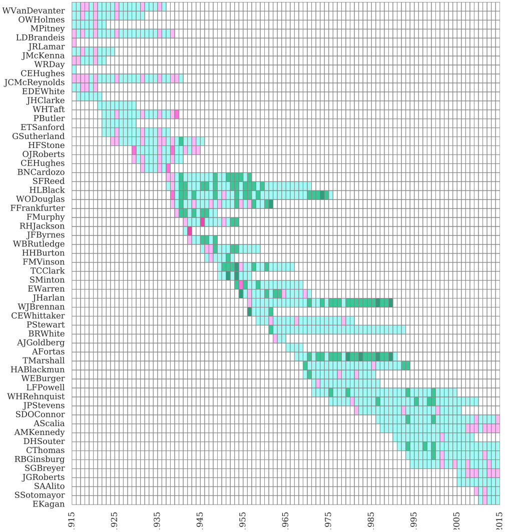
# *ex)* Predicting US Supreme Court behavior

**Random forest** classifier built on SCDB data[1] to predict the votes of Supreme Court justices:

$$Y \in \{\texttt{affirm}, \texttt{reverse}, \texttt{other}\}$$

**Result:** $70\%$ correct classifications

D. M. Katz, M. J. Bommarito II and J. Blackman. **A General Approach for Predicting the Behavior of the Supreme Court of the United States**. *arXiv.org*, arXiv:1612.03473v2, January 2017.



[1] http://supremecourtdatabase.org

# *ex)* Predicting US Supreme Court behavior

> *Not only have random forests proven to be "unreasonably effective" in a wide array of supervised learning contexts, but in our testing, random forests outperformed other common approaches including support vector machines [. . . ] and feedforward artificial neural network models such as multi-layer perceptron*
>
> — Katz, Bommarito II and Blackman (arXiv:1612.03473v2)

# Random forests

# Random forests

- ▲ Bagging can drastically improve the performance of CART!
- ▼ However, the $B$ bootstrapped dataset are *correlated*
  $\Rightarrow$ the variance reduction due to averaging is diminished.

**Idea:** De-correlate the $B$ trees by randomly perturbing each tree.

A **random forest** is constructed by bagging, but for each split in each tree only a ***random subset*** of $q \leq p$ inputs are considered as splitting variables.

Rule of thumb: $q = \sqrt{p}$ for classification trees and $q = p/3$ for regression trees.

# Random forest pseudo-code

**Algorithm** Random forest for regression

```
1. For $b = 1$ to $B$ *(can run in parallel)*
    (a) Draw a bootstrap data set $\widetilde{\mathcal{T}}$ of size $n$ from $\mathcal{T}$.
    (b) Grow a regression tree by repeating the following steps until a minimum node size is reached:
        i. Select $q$ out of the $p$ input variables uniformly at random.
        ii. Find the variable $x_j$ among the $q$ selected, and the corresponding split point $s$, that minimizes the squared error.
        iii. Split the node into two children with $\{x_j \leq s\}$ and $\{x_j > s\}$.
2. Final model is the average of the $B$ ensemble members,
    $$\widehat{y}_\star^{\text{rf}} = \frac{1}{B}\sum_{b=1}^{B} \widetilde{y}_\star^{b}.$$
```

# Random forests

**Recall:** For i.d. random variables $\{z_b\}_{b=1}^{B}$

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} z_b\right) = \frac{1-\rho}{B}\sigma^2 + \rho\sigma^2$$

The random input selection used in random forests:

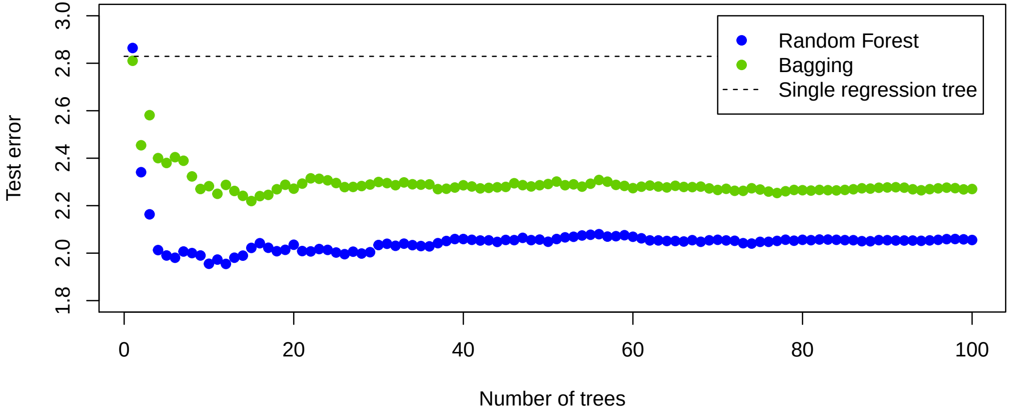
- ▼ increases the bias, but often very slowly
- ▼ adds to the variance ($\sigma^2$) of each tree
- ▲ reduces the correlation ($\rho$) of the trees

The reduction in correlation is typically the dominant effect ⇒ there is an overall reduction in MSE!

UPPSALA
UNIVERSITET

# *ex)* Toy regression model

For the toy model previously considered...



# Overfitting?

The complexity of a bagging/random forest model increases with an increasing number of trees $B$.

Will this lead to overfitting as $B$ increases?

No – more ensemble members ***does not*** increase the ***flexibility*** of the model!

**Regression case:**

$$\widehat{y}_{\star}^{\text{rf}} = \frac{1}{B}\sum_{b=1}^{B} \widetilde{y}_{\star}^{b} \to \mathbb{E}\left[\widetilde{y}_{\star} \mid \mathcal{T}\right], \qquad \text{as } B \to \infty,$$

where the expectation is w.r.t. the randomness in the data bootstrapping and input selection.

# Advantages of random forests

Random forests have several **computational advantages**:

- Embarrassingly parallelizable!
- Using $q < p$ potential split variables reduces the computational cost of each split.
- We ***could*** bootstrap fewer than $n$, say $\sqrt{n}$, data points when creating $\widetilde{\mathcal{T}}^b$ — very useful for “big data” problems.

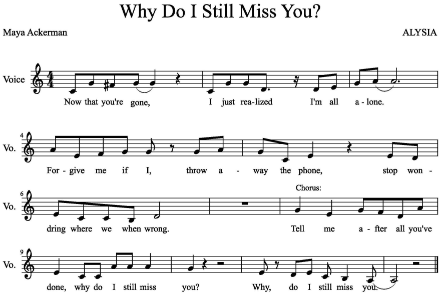
. . . and they also come with some other benefits:

- Often works well off-the-shelf – few tuning parameters
- Requires little or no input preparation
- Implicit input selection

# *ex)* Automatic music generation

**ALYSIA:** automated music generation using random forests.

- User specifies the lyrics
- ALYSIA generates accompanying music via
  - *rythm model*
  - *melody model*
- Trained on a corpus of pop songs.


Why Do I Still Miss You?
Maya Ackerman
ALYSIA
Voice
Now that you're gone, I just rea-lized I'm all a - lone.
4
Vo.
For - give me if I, throw a - way the phone, stop won -
Chorus:
6
Vo.
dring where we when wrong. Tell me a - fter all you've
9
Vo.
done, why do I still miss you? Why, do I still miss you.

https://www.youtube.com/watch?v=whgudcj82_I https://www.withalysia.com/

M. Ackerman and D. Loker. **Algorithmic Songwriting with ALYSIA**. *In: Correia J., Ciesielski V., Liapis A. (eds) Computational Intelligence in Music, Sound, Art and Design. EvoMUSART*, 2017.

# Boosting

# Boosting

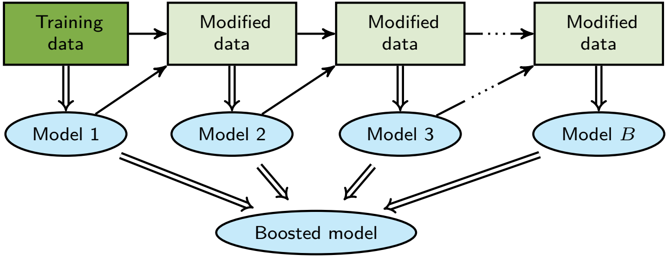
Even a simple (classification or regression) model can typically capture some aspects of the input-output relationship.

Can we then learn an ***ensemble*** of “weak models”, each describing some part of this relationship, and combine these into one “strong model”?

**Boosting:**

- ***Sequentially*** learns an ensemble of ***weak models***.
- Combine these into one ***strong model***.
- General strategy – can in principle be used to improve any supervised learning algorithm.
- A very successful idea!

# Boosting



The models are built **sequentially** such that each models tries to **correct the mistakes** made by the previous one!

# Boosting vs. bagging

| Bagging | Boosting |
|---|---|
| Learns base models in parallel | Learns base models sequentially |
| Uses bootstrapped datasets | Uses reweighted datasets |
| Does not overfit as $B$ becomes large | Can overfit as $B$ becomes large |
| Reduces variance but not bias<br>(suitable for models with low bias) | Primarily reduces bias!<br>(models with high bias are fine) |

Boosting does **not** require each base model to have low bias. Thus, a shallow classification tree (say, 4-8 terminal nodes) or even a tree with a single split (2 terminal nodes, a “stump”) is often sufficient.

# Binary classification

We will restrict our attention to binary classification.

- Class labels are $-1$ and $1$, i.e. $y \in \{-1, 1\}$.
- We have access to some (weak) base classifier, e.g. a classification tree.

*Note.* Using labels $-1$ and $1$ is mathematically convenient as it allows us to express a majority vote between $B$ classifiers $\widehat{y}^1(\mathbf{x})$, $\dots$, $\widehat{y}^B(\mathbf{x})$ as

$$\text{sign}\left(\sum_{b=1}^{B} \widehat{y}^b(\mathbf{x})\right) = \begin{cases} +1 & \text{if more plus-votes than minus-votes,} \\ -1 & \text{if more minus-votes than plus-votes.} \end{cases}$$

# Boosting procedure (for classification)

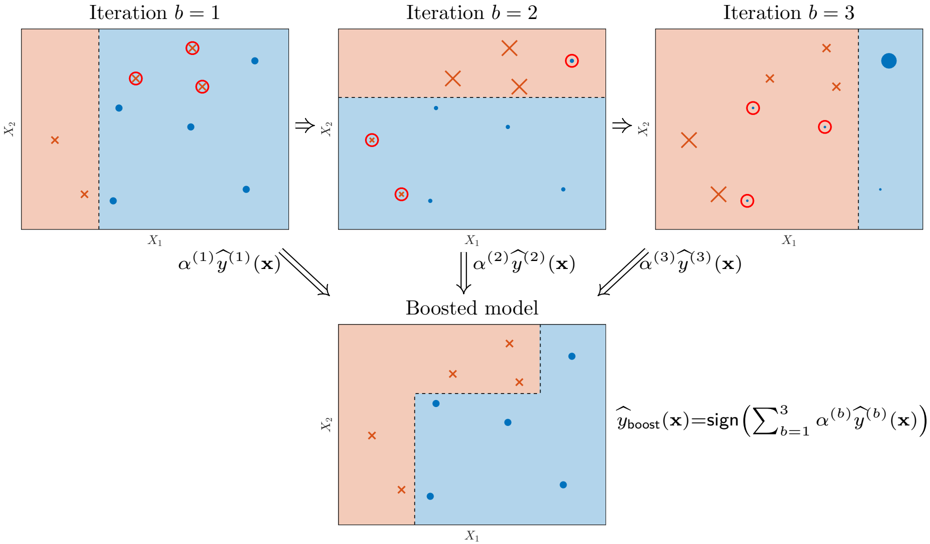
**Boosting procedure:**

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to $B$
   (a) Train a weak classifier $\widehat{y}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
   (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ *from* $\{w_i^b\}_{i=1}^n$*:*
      i. Increase weights for all points misclassified by $\widehat{y}^{(b)}(\mathbf{x})$.
      ii. Decrease weights for all points correctly classified by $\widehat{y}^{(b)}(\mathbf{x})$.

The predictions of the $B$ classifiers, $\widehat{y}^{(1)}(\mathbf{x}), \dots, \widehat{y}^{(B)}(\mathbf{x})$, are combined using a **weighted** majority vote:

$$\widehat{y}^B_{\text{boost}}(\mathbf{x}) = \text{sign}\left(\sum_{b=1}^{B} \alpha^{(b)} \widehat{y}^{(b)}(\mathbf{x})\right).$$

# *ex)* Boosting illustration


Iteration $b = 1$
Iteration $b = 2$
Iteration $b = 3$
$X_1$
$X_2$
$\Rightarrow$
$\alpha^{(1)}\widehat{y}^{(1)}(\mathbf{x})$
$\alpha^{(2)}\widehat{y}^{(2)}(\mathbf{x})$
$\alpha^{(3)}\widehat{y}^{(3)}(\mathbf{x})$
Boosted model

$$\widehat{y}_{\text{boost}}(\mathbf{x}) = \text{sign}\left(\sum_{b=1}^{3} \alpha^{(b)}\widehat{y}^{(b)}(\mathbf{x})\right)$$

# Important details

**Q1:** How do we reweight the data?

**Q2:** How are the coefficients $\alpha^{(1)}, \dots, \alpha^{(B)}$ computed?

# AdaBoost pseudo-code

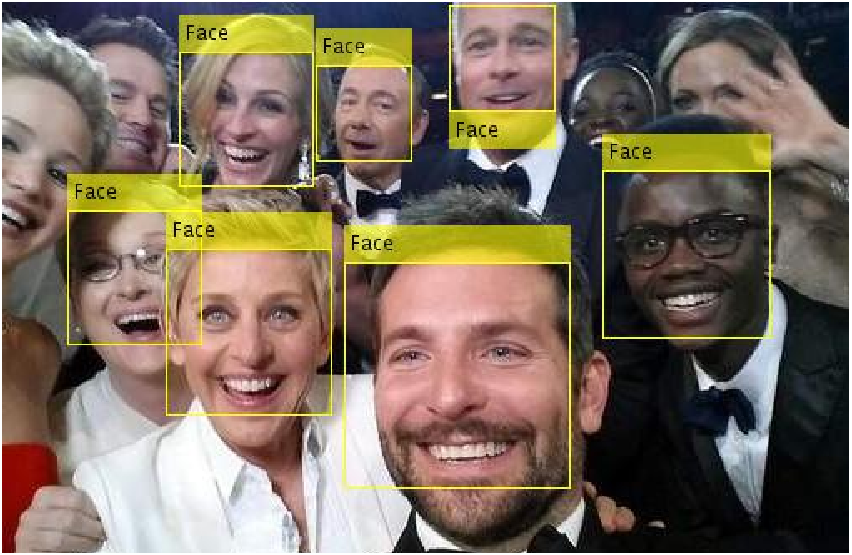
```
1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to $B$
    (a) Train a weak classifier $\widehat{y}^{(b)}(\mathbf{x})$ on the weighted training data $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
    (b) Update the weights $\{w_i^{b+1}\}_{i=1}^n$ from $\{w_i^b\}_{i=1}^n$:
        i. Increase weights for all points misclassified by $\widehat{y}^{(b)}(\mathbf{x})$.
        ii. Decrease weights for all points correctly classified by $\widehat{y}^{(b)}(\mathbf{x})$.
        i. Compute weighted classification errorCompute $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \widehat{y}^{(b)}(\mathbf{x}_i)\}$
        ii. Compute classifier "confidence"Compute $\alpha^b = 0.5 \log((1 - E_{\text{train}}^b)/E_{\text{train}}^b)$.
        iii. Compute new weightsCompute $w_i^{b+1} = w_i^b \exp(-\alpha^{(b)} y_i \widehat{y}^{(b)}(\mathbf{x}_i))$, $i = 1, \dots, n$
        iv. Normalize. Set $w_i^{b+1} \leftarrow w_i^{b+1} / \sum_{j=1}^n w_j^{b+1}$, for $i = 1, \dots, n$.
3. Output $\widehat{y}_{\text{boost}}^{(B)}(\mathbf{x}) = \text{sign}\left(\sum_{b=1}^B \alpha^{(b)} \widehat{y}^{(b)}(\mathbf{x})\right)$.
```

Y. Freund and R. E. Schapire. **Experiments with a New Boosting Algorithm**. *Proceedings of the 13th International Conference on Machine Learning (ICML)*. Bari, Italy, 1996.

2003 Gödel Prize

UPPSALA
UNIVERSITET

# *ex)* The Viola-Jones face detector



# A few concepts to summarize lecture 6

**CART:** Classification and regression trees. A class of nonparametric methods based on partitioning the input space into regions and fitting a simple model for each region.

**Recursive binary splitting:** A greedy method for partitioning the input space into “boxes” aligned with the coordinate axes.

**Gini index and deviance:** Commonly used error measures for constructing classification trees.

**Ensemble methods:** Umbrella term for methods that average or combine multiple models.

**Bagging:** Bootstrap aggregating. An ensemble method based on the statistical bootstrap.

**Random forests:** Bagging of trees, combined with random feature selection for further variance reduction (and computational gains).

**Boosting:** Sequential ensemble method, where each consecutive model tries to correct the mistakes of the previous one.

**AdaBoost:** The first successful boosting algorithm. Designed for binary classification.